AVR-8-bit-Mikrocontroller
Gruppe 200 - Einsetzen von AVR-Tools
Teil 206 - C-Compiler und AVR Studio



Hinweis

Externe Anschaltungen und Hardware-Erweiterungen werden in der **Gruppe 400 - ASM-Projekte** und in der **Gruppe 600 - C-Projekte** detailliert beschrieben.

Vorbemerkung

Nichts ist vollkommen - und nichts ist endgültig! So auch nicht dieses Tutorial! Deshalb bitte immer erst nach dem neuesten Datum schauen. Vielleicht gibt es wieder etwas Neues oder eine Fehlerbereinigung oder eine etwas bessere Erklärung. Wer Fehler findet oder Verbesserungen vorzuschlagen hat, bitte melden (**info@alenck.de**).

Immer nach dem Motto: Das Bessere ist Feind des Guten und nichts ist so gut, dass es nicht noch verbessert werden könnte.

Bild-, Beispiel-, Form- und Tabellen-Nummern sind nach folgendem Schema aufgebaut, damit bei Einfügungen/Löschungen nicht alle Nummern wieder geändert werden müssen (hier bunt dargestellt):

Darstellungsart Abschnitt-LfdNummer: Beschreibung	**allgemeines Schema**
• **Bild 5.1.4-02: Daten-Adress-Raum**	Benummerung eines Bildes
• **Beispiel 5.1.4-03: EEPROM-Speicherung**	Benummerung eines Beispiels
• **Form 5.1.3-01: Die `main`-Funktion**	Benummerung einer Formdarstellung
• **Tabelle 5.1.4-01: Schlüsselwörter vom CVAVR**	Benummerung einer Tabelle

Gravierende Änderungen gegenüber der Vorversion

1.
Völlig neue Strukturierung in **Gruppen** und **Teile**, um das Tutorial umfassend ordnen zu können. Die **Abschnitte** in den **Teilen** sind weitgehend erhalten geblieben.

Gruppenbezeichnung	Kurzbezeichnung
Gruppe 100: Technologie der AVR-8-Bit-Mikrocontroller	**Technologie**
Gruppe 200: Einsetzen von AVR-Tools	**Tools**
Gruppe 300: Arbeiten mit AVR-Assembler **3xx_Programm_yyyyy**	**ASM-Programmierung** **ASM-Programm-Beispiel**
Gruppe 400: AVR-ASM-Projekte **4xx_Projekt_yyyyy**	**ASM-Projekte** **ASM-Projekt-Bezeichnung**
Gruppe 500: CodeVisionAVR C-Compiler **5xx_Programm_yyyyy**	**C-Programmierung** **C-Programm-Beispiel**
Gruppe 600: AVR-C-Projekte **6xx_Projekt_yyyyy**	**C-Projekte** **C-Projekt-Bezeichnung**

xx steht für die laufende Nummer innerhalb des **Teils**, in dem das Programm bzw. das Projekt erscheint und **yyyyy** steht für die Programm- bzw. Projekt-Kurz-Bezeichnung.

2.
Notwendige Änderungen auf Grund Neuinstallation von **Windows 7**.

3.
Windows 7 machte eine Installation von **CodeVisionAVR V2.60** als Vollversion notwendig. Daraus leiten sich auch viele Änderungen im Detail für die C-Programmierung (**Gruppe 500**) ab.

4.
Neu-Installation von **AVR Studio Vers. 4.19** unter **Windows 7**

5.
Zur Demonstration des Tools **AVR Studio** ist in **Gruppe 200** eine Trennung in **Teil 205 - Assembler und AVR Studio** und **Teil 206 - C-Compiler und AVR Studio** vorgenommen worden.

6.
ASM- und **C-Projekte** werden jeweils in eigenen Gruppen gesammelt (**Gruppe 400** für Assembler- und **Gruppe 600** für C-Projekte).

6 CodeVisionAVR C-Compiler und AVR Studio

Anmerkungen:

1. Dieser **Teil 206** setzt eigentlich Programmierkenntnisse in der Programmiersprache C zur Erzeugung lauffähiger Programme mit dem CodeVisionAVR C-Compiler voraus. Da hier allerdings nur die Handhabung von AVR Studio und das in der **Gruppe 500** detailliert behandelten CodeVisionAVR C-Compilers beschrieben werden soll, werden die Schritte anhand eines fertigen kleinen Programms erläutert.

2. Von der Seite **http://lehrer.schule.at/Doblinger/files/AVR Studio 4 Tutorial.pdf** (die leider nicht mehr auffindbar ist) sind in diesem Abschnitt einige Anregungen entnommen worden.

6.1 CodeVisionAVR C-Compiler installieren

Die Installation der Version aus dem Download-Center vom www.cczwei.de des ATM18-Projektes ist nicht unter **Windows 7** lauffähig! (**Man beachte:** für alle Versionen von CodeVisionAVR C-Compiler heißen die Installationsprogramme **`setup.exe`**)

Auch ein neuer Versuch mit **CodeVisionAVR Evaluation V2.05.7a** von der **HP Info Tech-Homepage** http://www.hpinfotech.ro/html/download.htm führte leider nicht zum vollen Erfolg, da die Beispielprogramme unter den Einschränkungen der Demoversion nicht lauffähig sind.

Ein "Bettelbrief" an den Hersteller **HP Info Tech** mit Hinweis auf das ATM18-Projekt

http://www.cczwei-forum.de/cc2/thread.php?threadid=1797

hat dem Verfasser dann eine verbilligte Vollversion eingebracht. Inzwischen ist die **Version V2.60** verfügbar, die hier installiert wird. Aus verständlichen Gründen kann das Setup-Passwort, welches man beim Kauf erhält, hier nicht wiedergegeben werden, so dass das entsprechende Menü "ausgepunktet" dargestellt wird.

Nach dem Download steht die Datei **`setup.exe`** zur Verfügung. Bei der Installation wird man sogar gefragt, ob man Deutsch als Sprache wünscht. Allerdings wird der Wunsch später nicht mehr erfüllt!

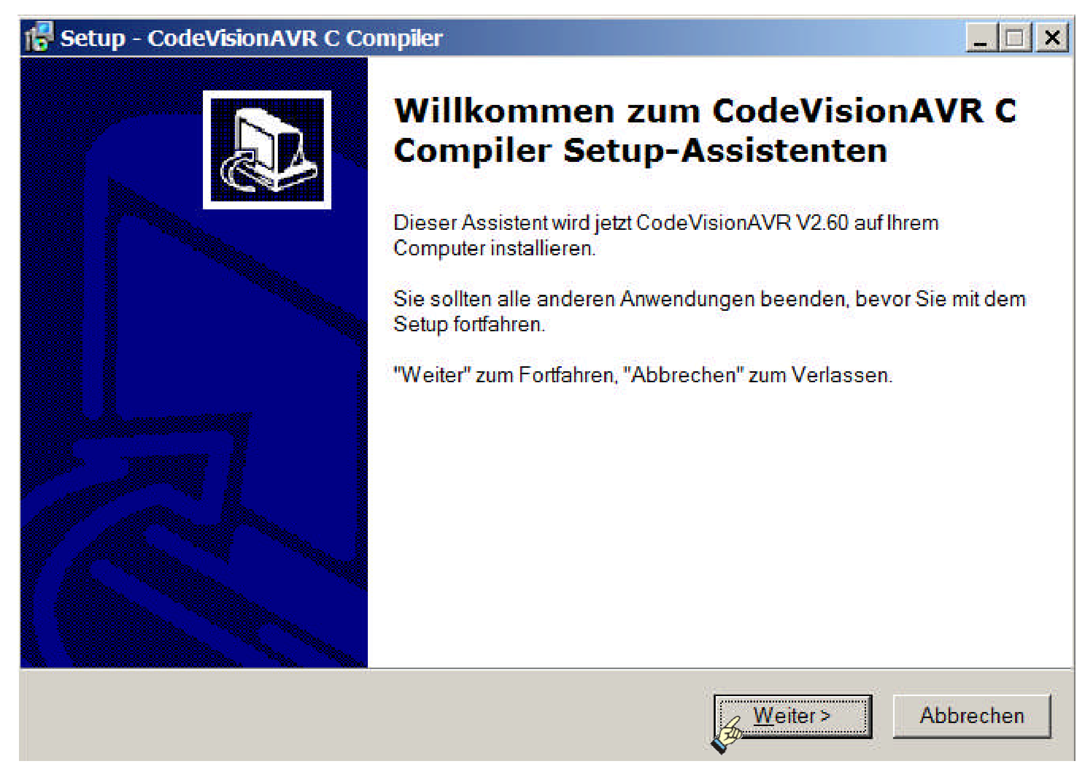


Bild 6.1-01: Begrüßungs-Fenster vom CodeVisionAVR C-Compiler

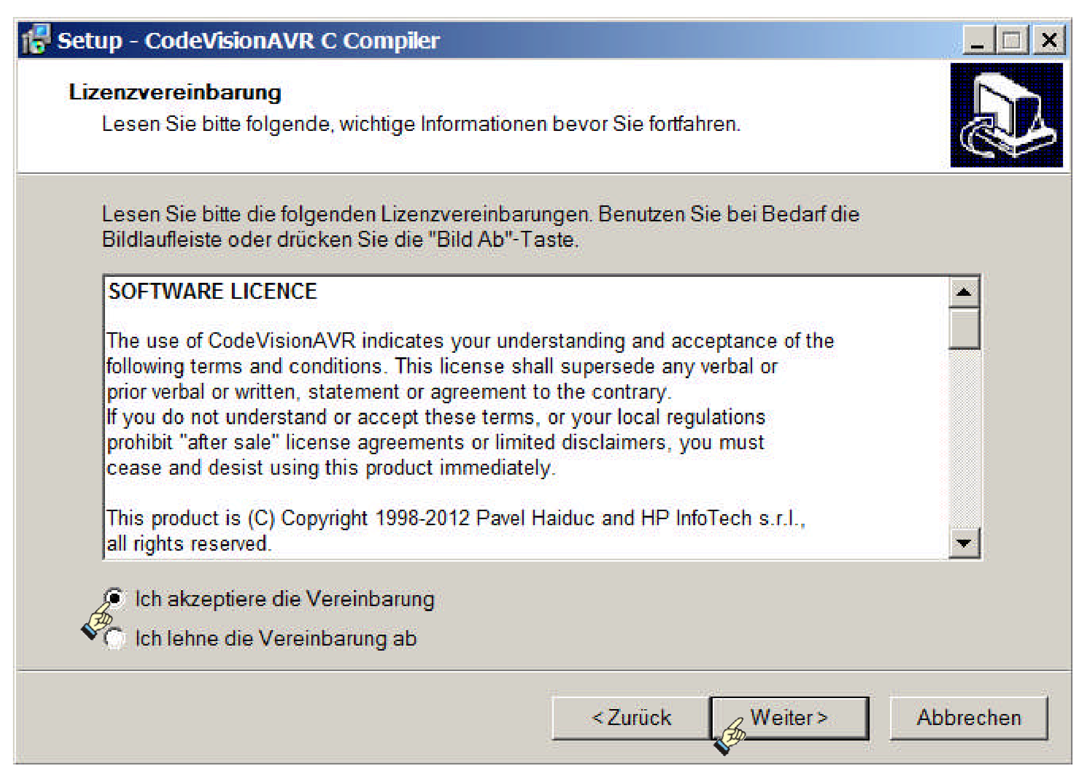


Bild 6.1-02: Lizenzvereinbarung akzeptieren

Die positive Antwort auf die obligatorische Anfrage nach dem Akzeptieren der Lizenzbestimmungen wird dann die Installation eingeleitet:

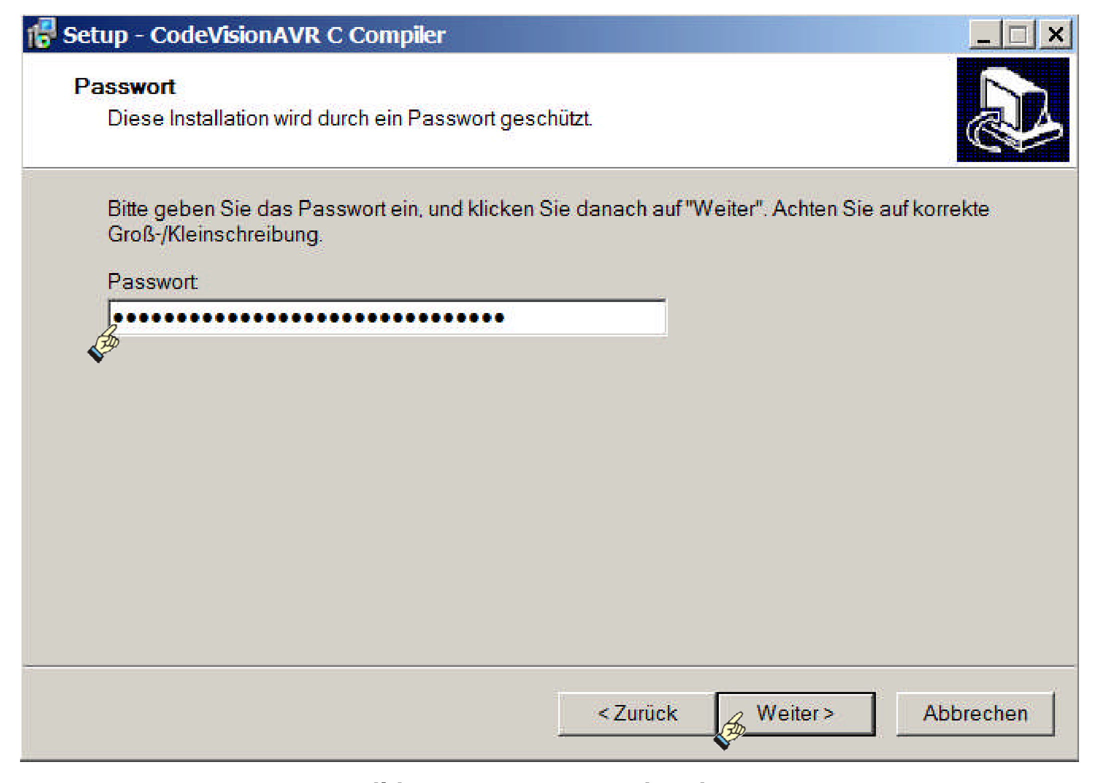


Bild 6.1-03: Passwort eingeben

Mit jeder neuen Programmversion ist ein neues Passwort beim Installieren einzugeben. Das Passwort und die Download-Adresse erhält man per E-Mail vom Hersteller **HP Info Tech s.r.l.**

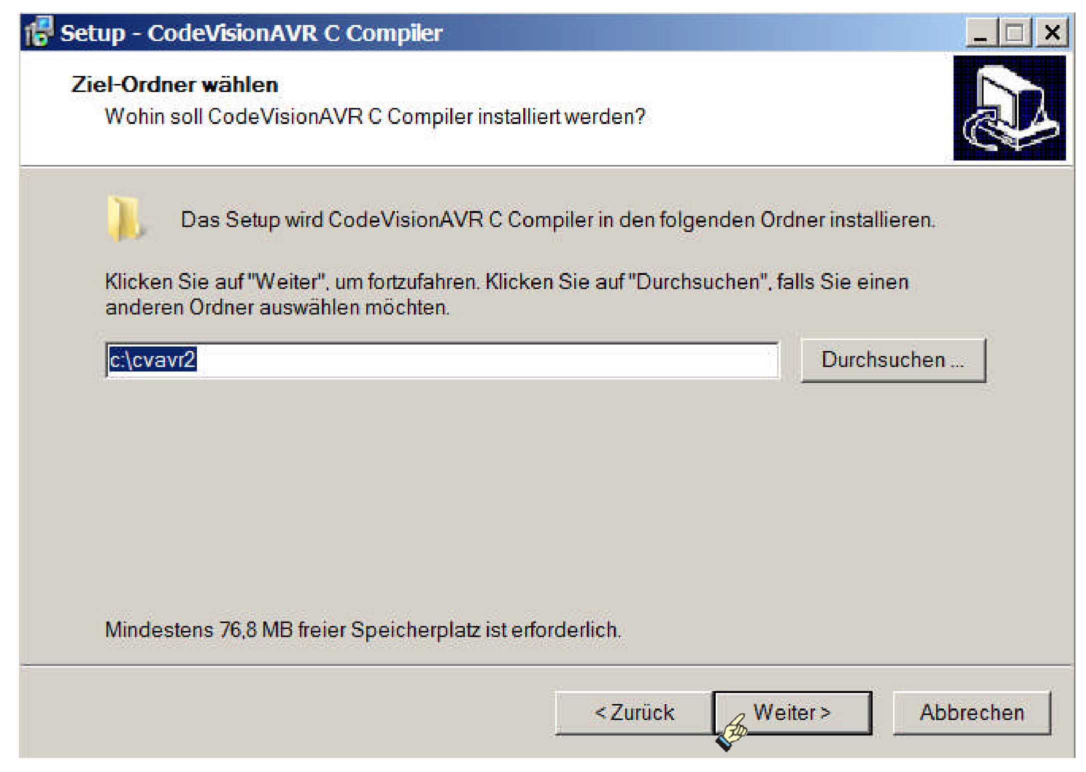


Bild 6.1-04: Installations-Ordner (sollte nicht verändert werden)

Der Ordner **c:\cvavr2** ist sehr wichtig und sollte nicht verschoben werden. Denn an diesem Ort werden auch die Bibliotheken abgespeichert.

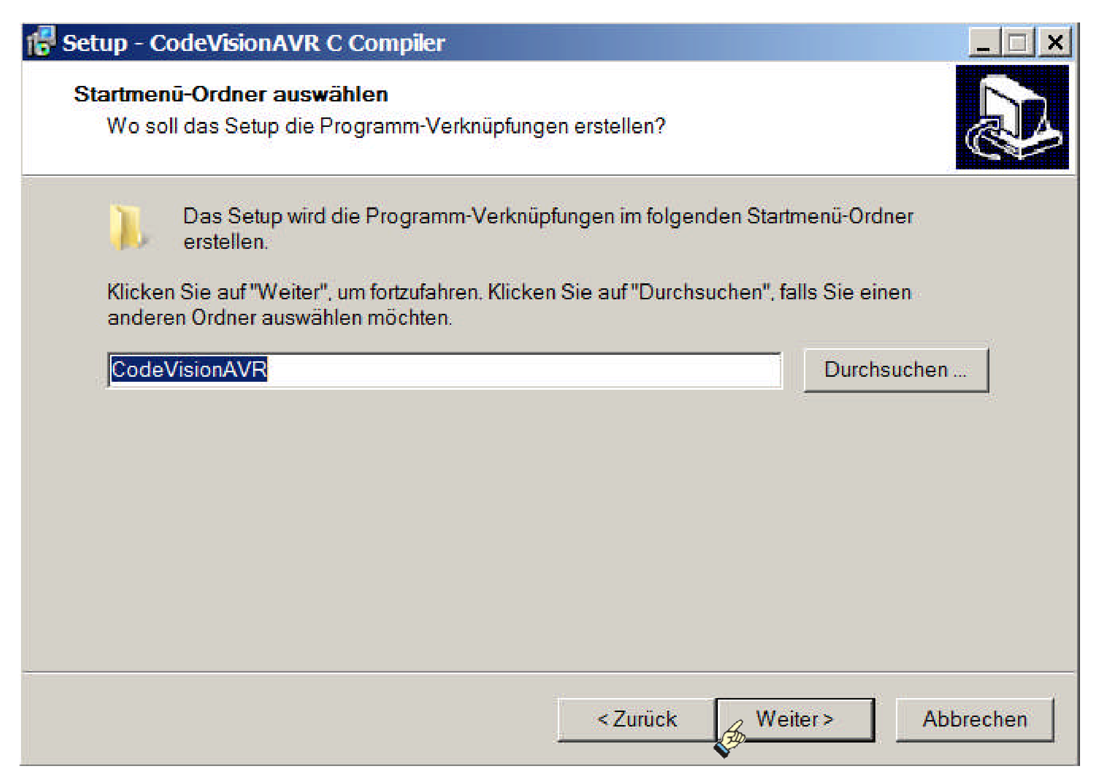


Bild 6.1-05: Startmenü-Ordner

Auch der Ordner für das Startmenü **CodeVisionAVR** sollte auf seinem vorgesehenen Platz belassen werden. Man findet die notwendigen Dateien schneller wieder (zum Beispiel):

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\CodeVisionAVR

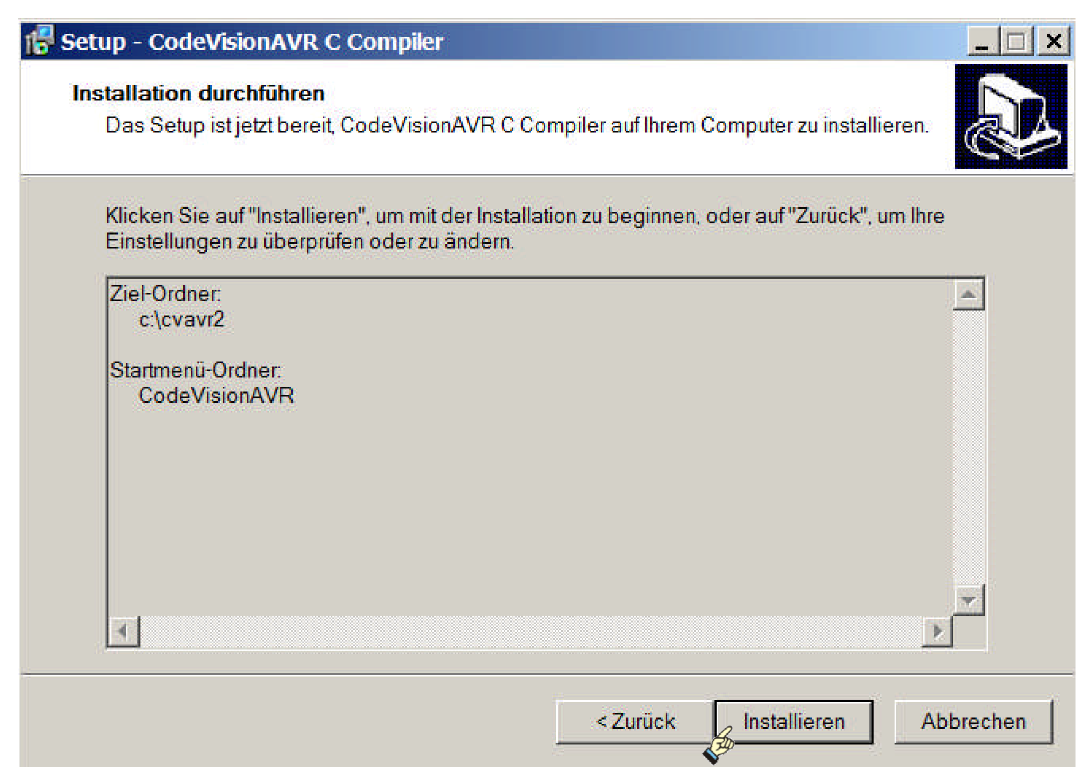


Bild 6.1-06: Installation durchführen

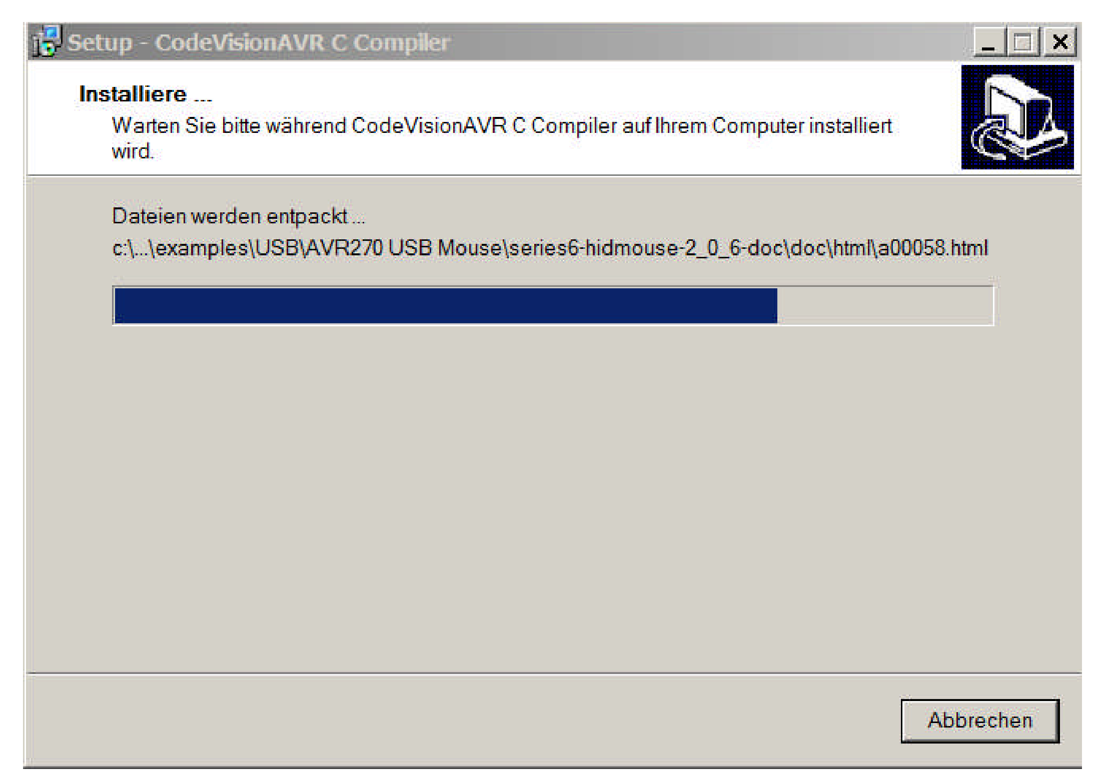


Bild 6.1-07: Installation läuft . . .

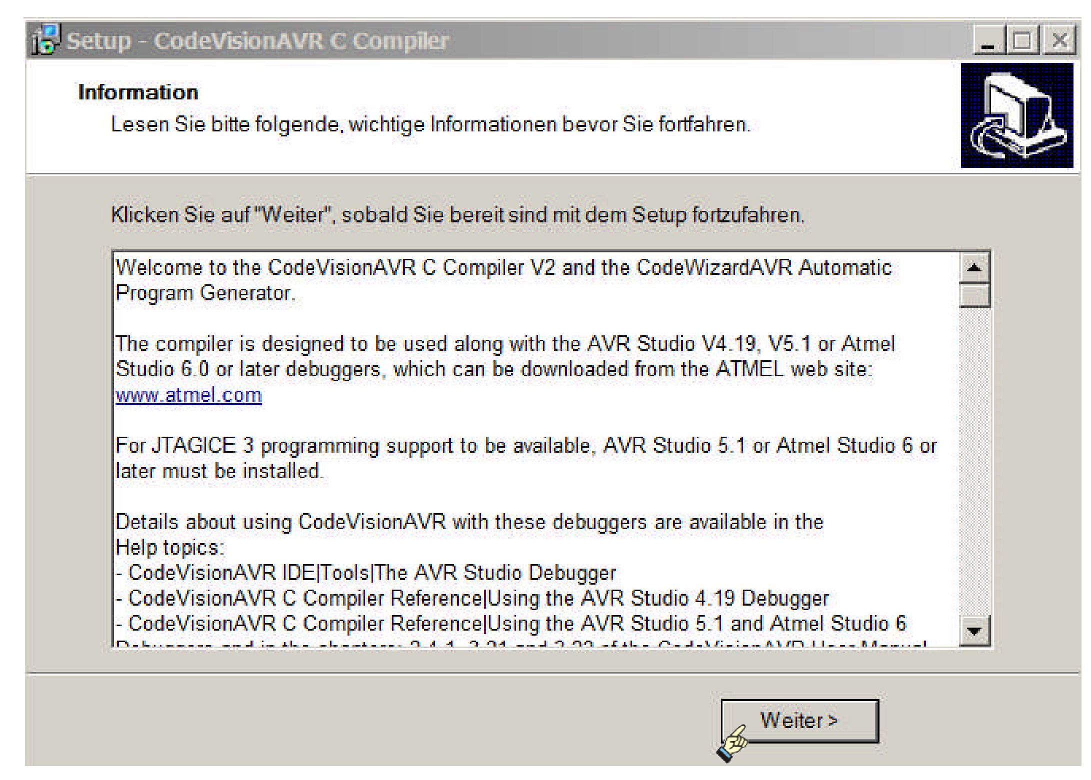


Bild 6.1-08: Setup - Wichtige Informationen

Text siehe weiter unten!

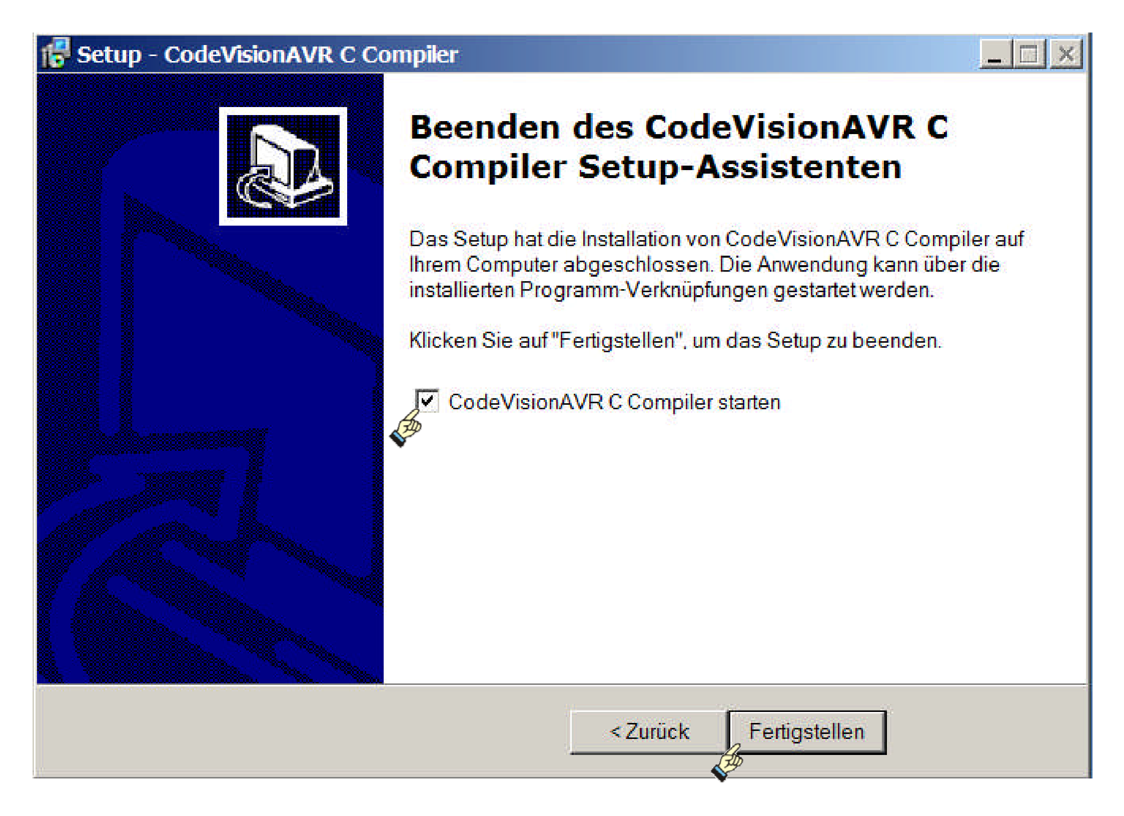


Bild 6.1-09: Installation ist abgeschlossen

Die neue Version unter Windows 7 wird gleich gestartet:



Bild 6.1-10: Man beachte, dass die "Version 2.60 Standard" keine Evaluation (Testversion) ist

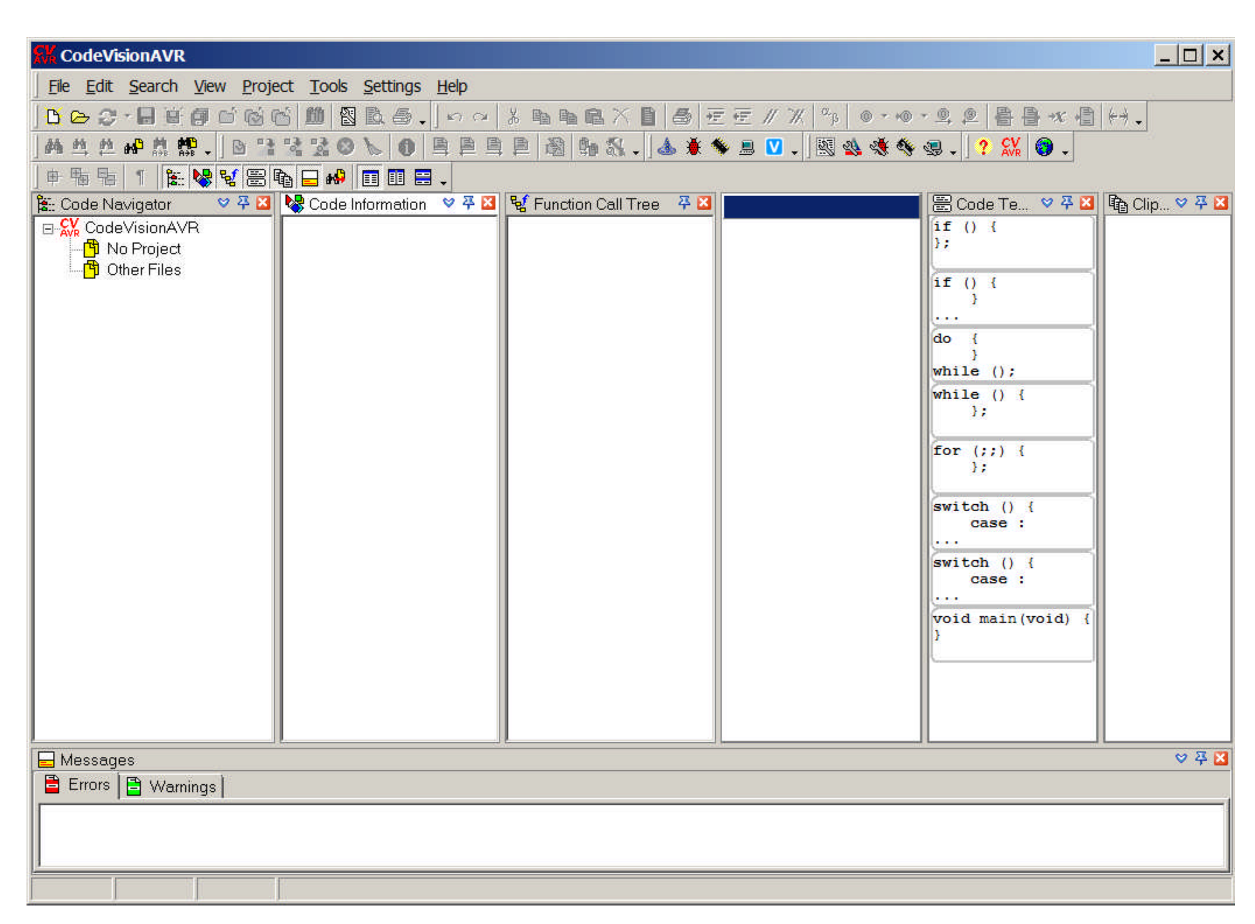


Bild 6.1-11: Erstes Bild nach der Installation (Bildvergrößerung)

Damit ist CVAVR installiert . . .

. . . und der erste Aufruf sollte ungefähr wie im **Bild 6.1-11** aussehen. Bei späteren Aufrufen, wenn schon mit dem CVAVR "gespielt" worden sein sollte, "erinnert" sich das Tool an die letzte Einstellung und gibt diese wieder. Wenn man nicht gerade an einem größeren Projekt über einen ausgedehnten Zeitraum arbeitet, bei dem diese Verfahrensweise sehr vorteilhaft ist, so wird man sicherlich bei einem neuen Projekt beginnen wollen. Das Eröffnungsfenster erhält man nach erstmaligem Aufruf oder allemal nach **File => Close All** - es sind jedoch ggf. einzelne Spalten ausgeblendet.

Zum CVAVR ist auch schon viel geschrieben worden, so dass es fast müßig ist, dem noch etwas hinzufügen zu wollen. Aber: Bei der Installation gibt es noch allerlei Kleinigkeiten, auf die es sich lohnt

hinzuweisen. Der lange Hinweis bei der Installation (**Bild 6.1-08: Setup**) ist schnell weggeklickt. Er ist aber in einigen Punkten sehr aufschlussreich und deshalb als Original-Zitat hier zum Nachlesen wiederholt:

Welcome to the CodeVisionAVR C Compiler V2 and the CodeWizardAVR Automatic Program Generator.

The compiler is designed to be used along with the AVR Studio V4.19, V5.1 or Atmel Studio 6.0 or later debuggers, which can be downloaded from the ATMEL web site: www.atmel.com

For JTAGICE 3 programming support to be available, AVR Studio 5.1 or Atmel Studio 6 or later must be installed.

Details about using CodeVisionAVR with these debuggers are available in the Help topics:

- CodeVisionAVR IDE|Tools|The AVR Studio Debugger
- CodeVisionAVR C Compiler Reference|Using the AVR Studio 4.19 Debugger
- CodeVisionAVR C Compiler Reference|Using the AVR Studio 5.1 and Atmel Studio 6 Debuggers and in the chapters: 2.4.1, 3.21 and 3.22 of the CodeVisionAVR User Manual.

1. Installing CodeVisionAVR

If you will use the Compiler under Windows 2000, XP, Vista or Windows 7 you must first install and run it with Administrator privileges. On subsequent runs under Windows 2000 or XP you may also have Power User privileges.

After installing under Windows 7, right click on the CodeVisionAVR icon and select **Properties|Shortcut|Advanced** and check the **Run as Administrator** check box.

For installing under Windows Vista, the following steps should be performed, having an **Administrator** account:

- the Windows **Vista User Account Control** must be disabled using the steps below:

  - press the **Windows** button and click on the **Control Panel** link
  - in the **Control Panel** window, under the **User Accounts and Family Safety**, click on the **Add or remove user accounts** link
  - a new window will open, click on the **Go to the main User Accounts page** link located at the bottom of the window
  - a new window will open, click on the **Turn User Account Control on or off** link located at the bottom of the window
  - in the new window that will open, uncheck the **Use User Account Control (UAC) to help protect your computer** check box and press the **OK** button to confirm
  - the computer will have to be restarted for this setting to become effective

- run the CodeVisionAVR setup.exe installer
- after the installation is complete, right click on the **CodeVisionAVR** icon on the desktop
- select **Properties** in the popup menu that will open
- a new window called **CodeVisionAVR Properties** will open
- select the **Compatibility** tab
- check the **Privilege Level|Run this program as an administrator** check box and press the **OK** button to confirm.

2. New features/bug fixes in CodeVisionAVR V2.60

2.1 Compiler

- Added support for the Ilitek ILI9325 graphic TFT LCD controller (only for Advanced license).
- Improved the library for the Solomon Systech SSD1289 graphic TFT LCD controller.
- Modified the alphanumeric LCD library (alcd.h) to be compatible with the Novatech NT3881 controller.

2.2 CodeWizardAVR

- Added the possibility to automatically set all the timer configuration registers for user specified operating mode, period and output duty cycle(s).
- Fixed: the Timer5 OC5A, OC5B, OC5C outputs were assigned to non-existent PORTM, instead of PORTL, for the ATmega640/1280/2560 chips.
- Fixed: the OC4A, OC4B, OC4C OC5A, OC5B, OC5C outputs are not present for the ATmega1281/2561 chips.
- Fixed: the ADTS0..3 bits in the SFIOR or ADCSRB registers were not set correctly, when using one of the ADC auto-trigger modes.
- Fixed: the ADC auto-trigger modes were missing for the ATmega32 chip.
- Fixed: USART code generated for the ATmega64 chip, used UMSEL00..UMSEL11 bit names instead of UMSEL0 and UMSEL1 in the UCSR0C and UCSR1C registers.
- Fixed: ADC code generated for the ATmega64 chip, used the ADATE bit name, which is not present in the ADCSRA register of this chip.

2.3 Chip Programmer

- Added check for conflicts between the 'EEPROM|Program' and 'Preserve EEPROM' options.
- Fixed error for restoring EEPROM contents after chip erase, when the 'Preserve EEPROM' option was enabled and

Atmel Studio 6 was set as debugger.

3. New features/changes in CodeVisionAVR V2.60, not available in old V1.25.x compilers

3.1 Compiler

- ANSI C compatible C front-end
- the bool data type was added (stdbool.h)
- the @ operator can now be used with variables located in EEPROM too
- the & unary operator can now be applied to I/O registers declared using the sfrb and sfrw keywords. It will return a pointer to the RAM address where these registers are mapped.
- there is no need to specify the 'flash' or 'eeprom' memory attribute during structure or union type definition. Therefore the same structure or union data type can be easily located in any memory area during variable declaration.
- the compiler now makes distinction between the 'const' type qualifier and the 'flash' memory attribute. In order to maintain compatibility with V1.25.x projects, the Project|Configure|C Compiler|Code Generation|Store Global Constants in FLASH Memory must be checked. If this option is not checked, identifiers declared with the 'const' type qualifier will be placed in RAM.
- the preprocessor does not use the old 'funcused' directive, when it is found it is always evaluated to 1
- added the #message preprocessor directive
- the abs, cabs, labs and fabs functions were moved from the math.h header to the stdlib.h header
- improved error and warning checking
- improved linker: only the modified C source files are compiled when the Project|Build command is executed
- improved COFF object file generator
- improved code optimizer
- enhanced libraries, including MMC/SD/SD HC FLASH Memory Card and FAT support
- I/O registers bits definitions were added to the device header files.
  For projects created with prior versions, these definitions are not enabled by default.
  In order to enable them the *Project|Configure|C Compiler|Code Generation|Preprocessor|Include I/O Registers Bits Definitions* option must be activated.
  For newly created projects, this option is enabled by default.
- in order to eliminate naming conflicts with I/O registers bits definitions, the names of the **SPCR0, SPDR0, SPSR0** registers were changed to **SPCR, SPDR, SPSR** in the following header files: mega164.h, mega324.h, mega644.h, mega644p.h.
- in order to eliminate naming conflicts with I/O registers bits definitions, the names of the **PCINT0, PCINT1, PCINT2, PCINT3** interrupt vectors were changed to **PC_INT0, PC_INT1, PC_INT2, PC_INT3** in the following header files: 90usb1286.h, 90usb1287.h, 90usb646.h, 90usb647.h, 90usb162.h, 90usb162.h, mega1280.h, mega1281.h, mega2560.h, mega2561.h, mega640.h, mega1284p.h, mega162.h, mega164.h, mega165.h, mega168.h, mega168p.h, mega169.h, mega324.h, mega325.h, mega325p.h, mega3250.h, mega3250p.h, mega328p.h, mega329.h, mega329p.h, mega3290.h, mega3290p.h, mega406.h, mega48.h, mega48p.h, mega644.h, mega644p.h, mega645.h, mega6450.h, mega649.h, mega6490.h, mega88.h, mega88p.h, tiny10.h, tiny13.h, tiny24.h, tiny25.h, tiny44.h, tiny45.h, tiny48.h, tiny84.h, tiny 85.h, tiny88.h, tiny261.h, tiny461.h, tiny861.h, tiny2313.h.
  If these interrupt vectors are used in your programs, their names must be updated.
- in order to eliminate naming conflicts with I/O registers bits definitions, the names of the **INT0...INT7** interrupt vectors were changed to **EXT_INT0...EXT_INT7** in the following header files: mega1280.h, mega1281.h, mega2560.h, mega2561.h, mega640.h, mega1284p.h, mega164.h, mega324.h, mega644.h, mega644p.h.
  If these interrupt vectors are used in your programs, their names must be updated.
- removed the limitation: when a literal char string enclosed between quotes, is passed as an argument to a function parameter of pointer to char, the pointer now can point to any memory type: RAM, EEPROM or FLASH, not only FLASH like in previous versions. The CodeVisionAVR C Compiler Reference|Constants Help topic and User Manual chapter were updated to reflect this change.

3.2 IDE

- completely redesigned text Editor with improved syntax highlighting for both C and AVR assembly
- added function parameters and global structure/union members auto complete
- improved code folding
- code folding state can be saved and restored
- bookmarks state can be saved and restored
- added automatic matching brace highlighting when the user places the cursor before the brace
- added automatic file saving at specified time interval
- the Code Navigator was redesigned and displays project information in a more logical way
- an additional Code Information tab is present after a project build. It displays detailed information about the included header files, preprocessor macro definitions, type definitions, global/static variable declarations, memory allocation and function definitions.
- added the Functions Call Tree tab in the Navigator
- the new Project|Configure|Files|Output Directories allows to specify in which directories the files generated by the compiler/linker will be placed.
- improved toolbar

It is important to note that in CodeVisionAVR V2 the .asm, .hex, .eep, .cof files created by the compiler have the name of the project .prj file.

3.3 CodeWizardAVR

- Generates code using symbolic names for peripheral configuration registers' bits.
- Improved user interface.

4. New ATxmega chips support in CodeVisionAVR

- the Standard C I/O Functions use by default the USARTC0. This can be changed by defining the _ATXMEGA_USART_ macro as described in the corresponding Help topic.
- the SPI functions use by default the SPIC controller on PORTC. This can be changed by defining the _ATXMEGA_SPI_ and _ATXMEGA_SPI_PORT_ preprocessor macros as described in the corresponding Help topic.
- the RTC functions do not yet support the ATxmega chips.

5. Example programs

The Compiler is supplied with the following example programs:

- ATxmega chips in \EXAMPLES ATxmega
- Graphic LCDs in \EXAMPLES\Graphic LCD
- SD FLASH Memory Cards access in \EXAMPLES\SDCARD
- USB in \EXAMPLES\USB
- AVR 109 Bootloader
- AES Encryption-Decryption
- "ATmega8535 ADC on the STK500" in \EXAMPLES\ADC8535
- "Real Time Clock with ATmega103, Atmel Application Note AVR134" in \EXAMPLES\AVR134
- "Calling functions written in assembler from C" in \EXAMPLES\C_ASM
- "Moving LED" in \EXAMPLES\LED
- "Accessing the EEPROM" in \EXAMPLES\EEPROM
- "LCD Demo" in \EXAMPLES\LCDDEMO
- "Definition of user characters in LCDs" in \EXAMPLES\LCDCHAR
- "LM75 Thermometer" in \EXAMPLES\THERM75
- "LCD Thermometer" in \EXAMPLES\THERMLCD
- "DS1820/DS18S20 Multipoint thermometer" in \EXAMPLES\DS1820
- "DS1990 Serial Number iButton" in \EXAMPLES\DS1990
- "AD7896 Digital voltmeter" in \EXAMPLES\SPI
- "MAX1241 Digital voltmeter" in \EXAMPLES\MAX1241
- "4x4 Keypad" in \EXAMPLES\KEYPAD
- "Simple multifile project" in \EXAMPLES\MULTFILE
- "Redirecting the output of standard C I/O functions to USART0, USART1 and LCD for an ATmega128 chip" in \EXAMPLES\2USART_LCD
- Tiny5 Level Meter
- uIP_Crumb644-Net_CVAVR web server
- TWI Master-Slave.

After starting CodeVisionAVR, execute the File|Open menu command and load the Project (*.prj) file for the example you like. Then execute the Project|Build All menu command to compile and link the project.

Take some time and carefully read the Help.

Auch C-Programme werden ausschließlich in Form von Projekten bearbeitet, so dass jetzt an die Bildung eines C-Projektes herangegangen werden soll.

6.2 Erzeugen eines C-Projektes

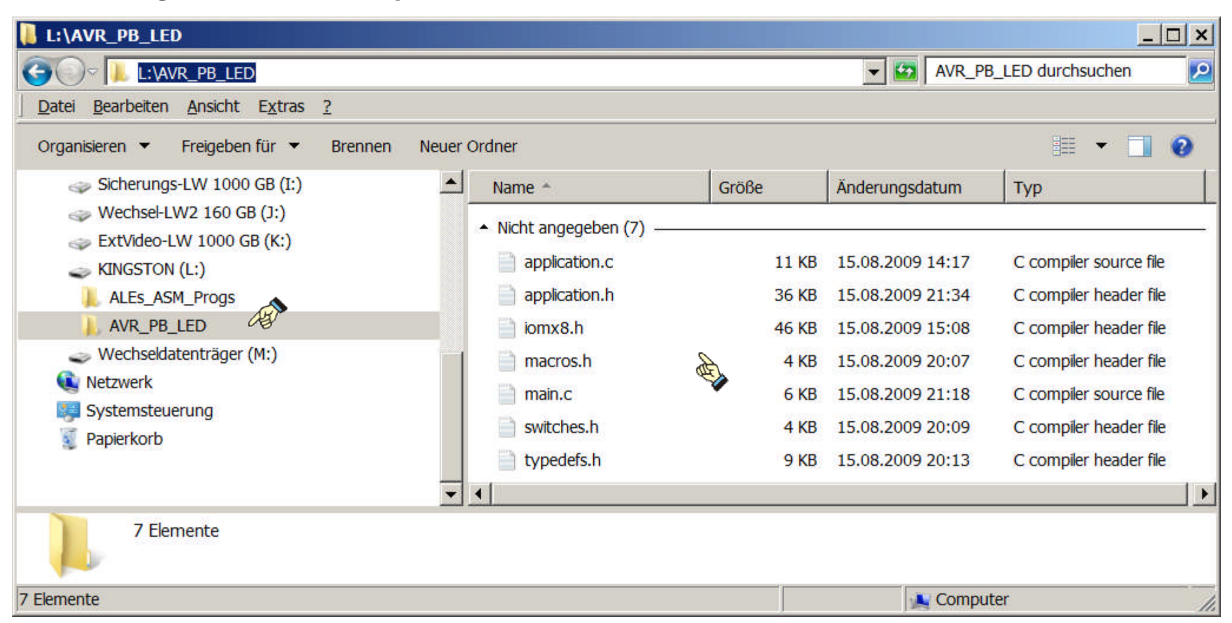


Bild 6.2-01: Einen neuen Projekt-Ordner mit (vorhandenen) Quell-Dateien anlegen

Zur Demonstration einer Projekt-Generierung wird ein neuer Projekt-Ordner **AVR_PB_LED** angelegt, in dem alle fertigen ***.c**- und ***.h**-Dateien des fertigen Projektes **AVR_PB_LED** übertragen werden.

Siehe: **Gruppe 600 - AVR C-Projekte, Teil 601 - AVR_PB_LED**
Einfache Beschaltung von LEDs und Tastern, der Quell-Code steht unter **601_Projekt_AVR_PB_LED**

Der Ordner-Name wurde vom Namen des bestehenden Projektes entlehnt. Dieser Ordner und die folgenden Schritte dienen gleichermaßen zur Demonstration, wie man als "Neuling" fertig erstellte Projekte zum Testen benutzen kann. Wenn man ein neues Projekt beginnt, fängt man ja bei "Adam und Eva" an, d.h. man legt einen neuen Projekt-Ordner an, übernimmt **oder** erstellt darin die Quell-Dateien und marschiert dann durch die hier beschriebenen Instanzen.

6.2.1 Ein neues Projekt beginnen

Oben wurde das Compiler-Tool **CodeVisionAVR** installiert; jetzt wird das erste Projekt (wenn auch nur entlehnt) durch Start dieses Tools erzeugt:

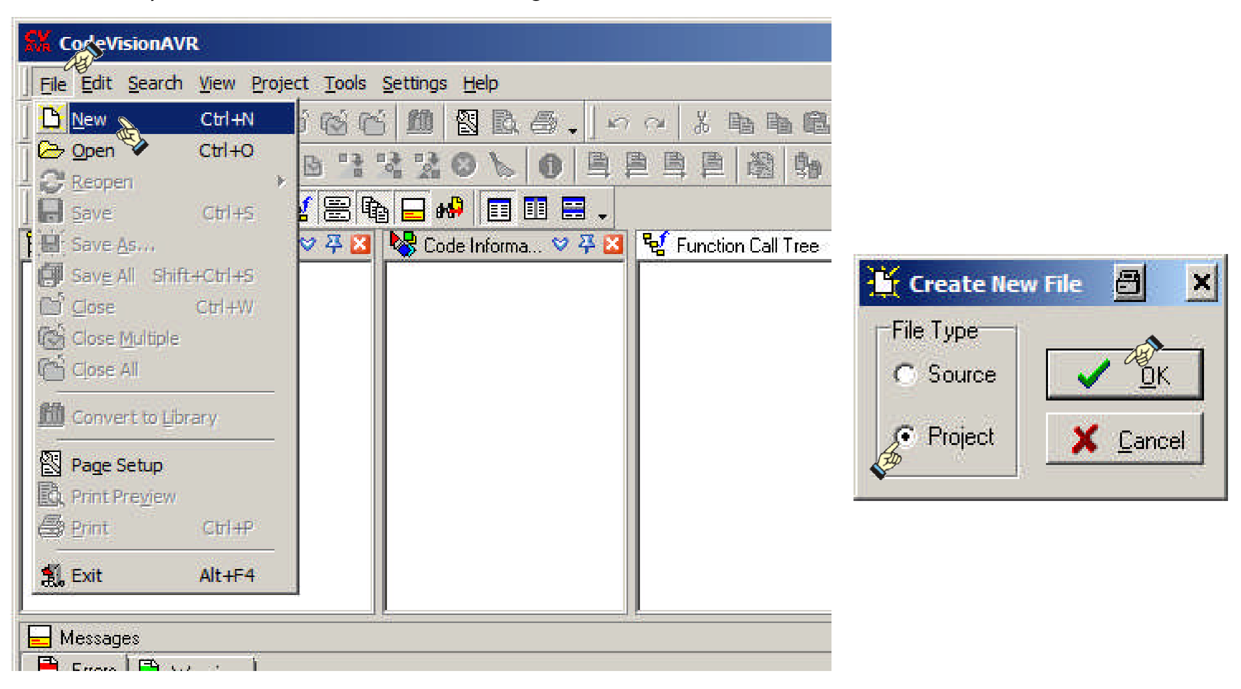


Bild 6.2.1-01: CVAVR für ein neues Projekt starten

File => New => Create New File => Project => OK

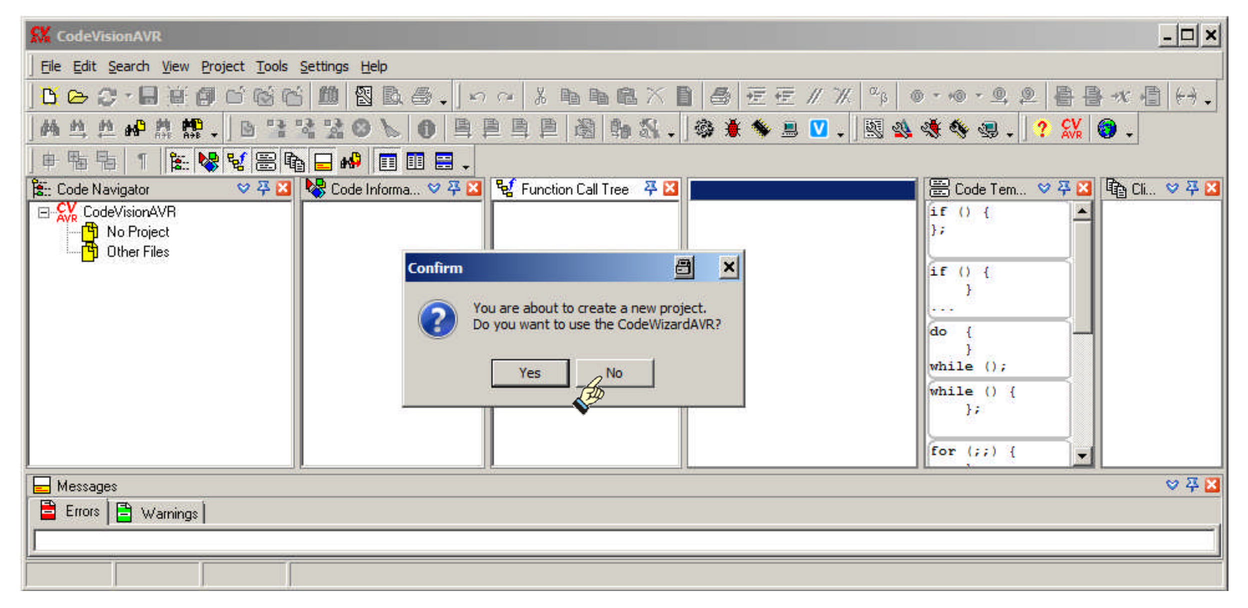


Bild 6.2.1-02: Anfrage, ob der CodeWizardAVR benutzt werden soll

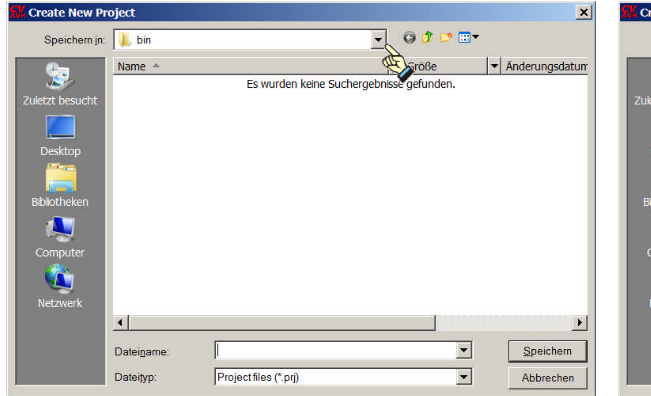
Es soll der **CodeWizardAVR** nicht verwendet werden: **=> No**; es meldet sich das Fenster **Create New Project**. Dieses verweist ggf. zunächst auf den Pfad der CVAVR-Installation:

`C:\cvavr2\bin`

weil der Compiler-Hersteller offenbar davon ausgeht, dass alle Projekte hier konzentriert werden sollen. Da aber das Projekt in einem eigens dafür vorgesehenen Ordner angelegt werden soll, müssen hier neue Einstellungen vorgenommen werden.

Im Feld **Speichern in:** wird der neu eingerichtete Ordner gesucht.

Hier: **`AVR_PB_LED`** **=> Speichern**



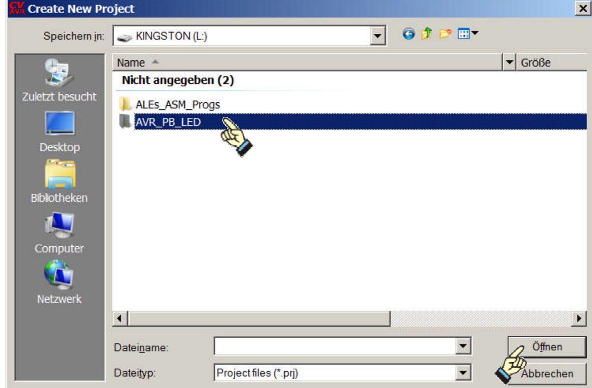


Bild 6.2.1-03: Create New Project **Bild 6.2.1-04: Ordner `AVR_PB_LED` suchen**

Klick auf und KINGSTON (L) wählen **AVR_PB_LED markieren => Öffnen**

Jetzt muss noch der Dateiname für das **Project file (`*.prj`)** benannt werden. Sinnvoll ist es, den gleichen Namen wie für den Projekt-Ordner zu wählen - nämlich **`AVR_PB_LED`**. Nur diese Datei braucht später aufgerufen zu werden, um Änderungen im Projekt vorzunehmen. Sie soll wie folgt abgelegt werden (hier beispielhaft auf dem USB-Stick **`L:`**):

`L:\AVR_PB_LED\AVR_PB_LED.prj`

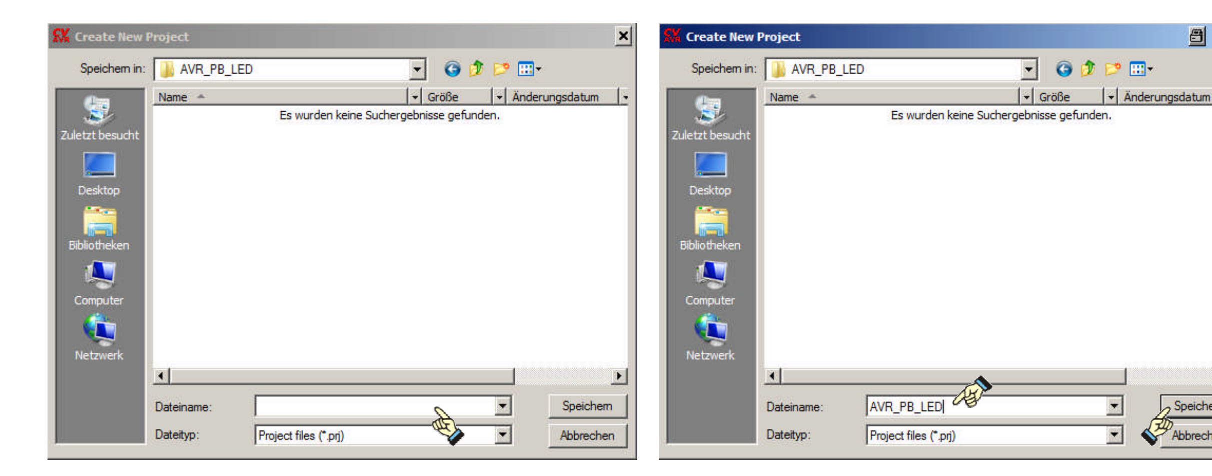


Bild 6.2.1-05: Es wird eine *.prj-Datei angefordert **Bild 6.2.1-06: Benennen der *.prj-Datei**

Es öffnet sich zusätzlich das Fenster **Configure Project AVR_PB_LED.prj** mit der Aufforderung, dem Projekt nun die Quell-Dateien hinzuzufügen (besser: bekannt zu machen, da sie sich ja bereits im Projekt-Ordner befinden).

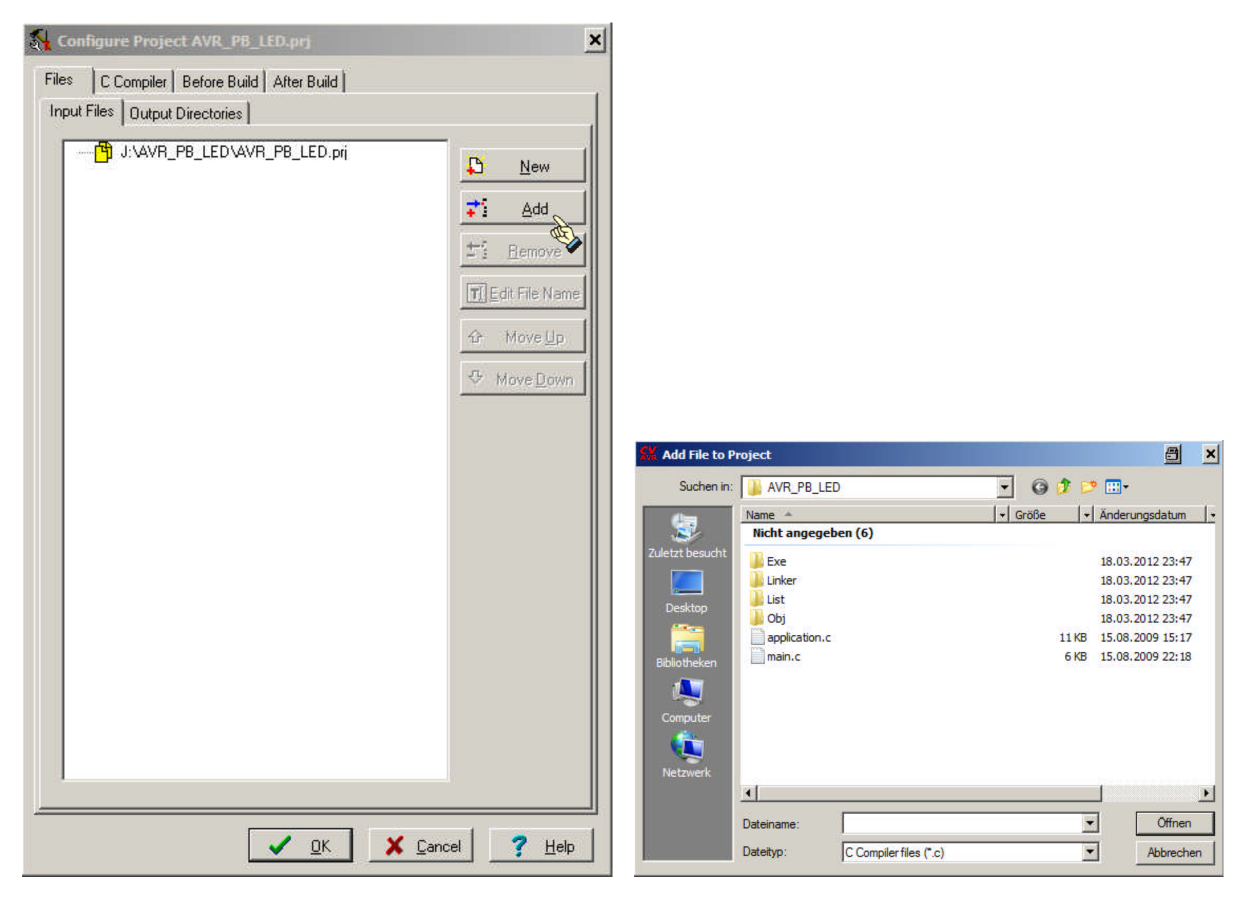


Bild 6.2.1-07: Configure Project Part 1 - Input Files **Add =>** **Bild 6.2.1-08: Neue Projekt-Struktur**

Im Hauptfenster von CVAVR sieht man bereits, dass auf der linken Seite im **Code Navigator** der Projekt-Name **AVR_PB_LED** und in der Kopfzeile der Name der Projekt-Datei **AVR_PB_LED.prj** eingetragen sind.

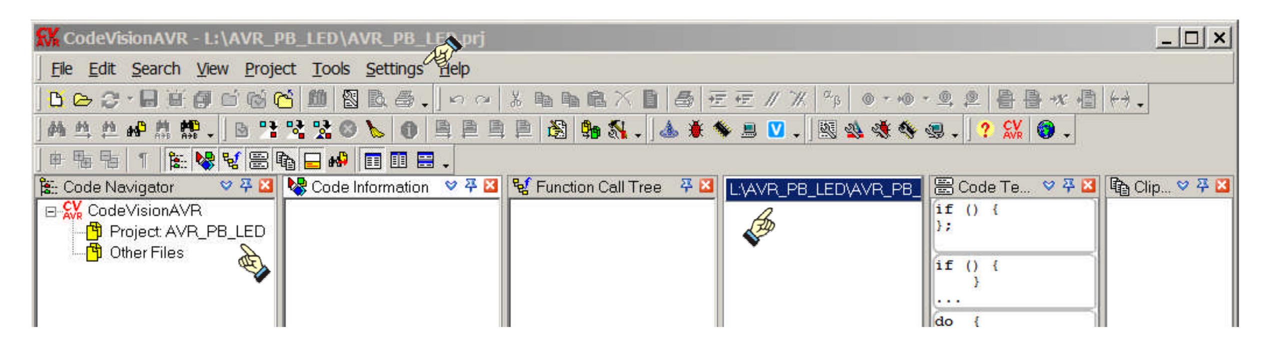


Bild 6.2.1-09: Hauptfenster von CVAVR

Jetzt müssen alle **C**-Dateien zum Kompilieren und Binden dem Projekt bekannt gemacht werden. Das Menü lässt automatisch nur **C**-Dateien zu und weist damit bereits darauf hin, dass nur die **C**-Dateien den anfänglichen Start bestimmen. Die Header-Dateien werden erst während der Kompilierung (genauer: während des Preprozessor-Laufs) aus den **C**-Dateien heraus aufgerufen und in das Coding eingefügt.

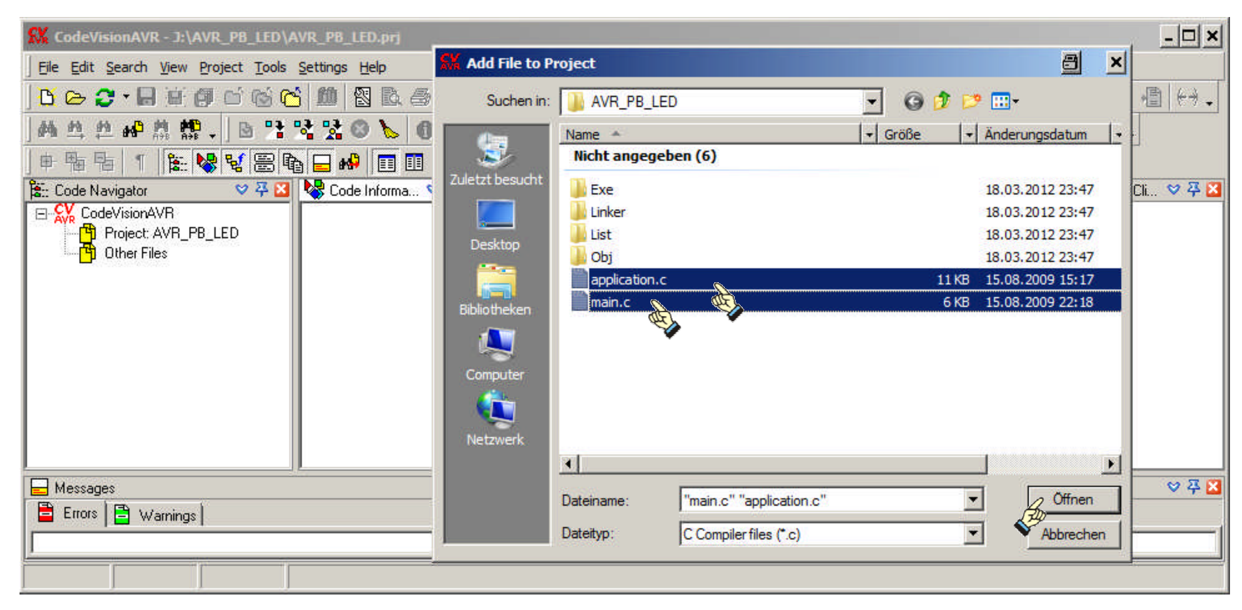


Bild 6.2.1-10: C-Dateien dem Projekt bekannt machen

Alle **C**-Dateien auswählen **=> Öffnen** und . . . . **OK im Configure Project-Menü**

Achtung: Der Vorgang dauert eine Weile - Bitte nicht die Geduld verlieren!

Noch mal der Reihe nach (wegen der Überschneidung der Menüs):

Bild 6.2.1-06: Benennen der *.prj-Datei: AVR_PB_LED.prj => Speichern

Bild 6.2.1-09: Hauptfenster von CVAVR - zur Übersicht

Bild 6.2.1-07: Configure Project - Part 1 (das Menü bleibt zunächst noch sichtbar) **=> Add**

Bild 6.2.1-08: Neue Projekt-Struktur

Bild 6.2.1-10: C-Dateien dem Projekt bekannt machen => *.C-Dateien markieren => Öffnen

Bild 6.2.1-07: Configure Project => OK

Durch **OK** im **Configure Project** werden alle ***.c**-Dateien in das Projekt übernommen.

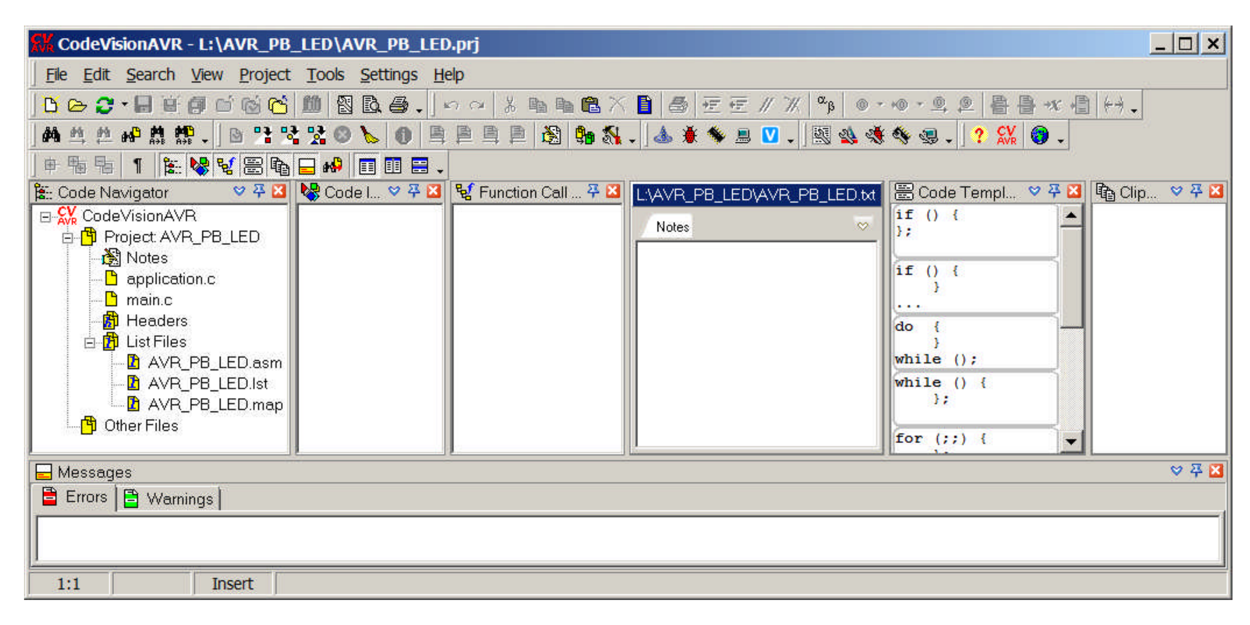


Bild 6.2.1-11: Hauptfenster von CVAVR

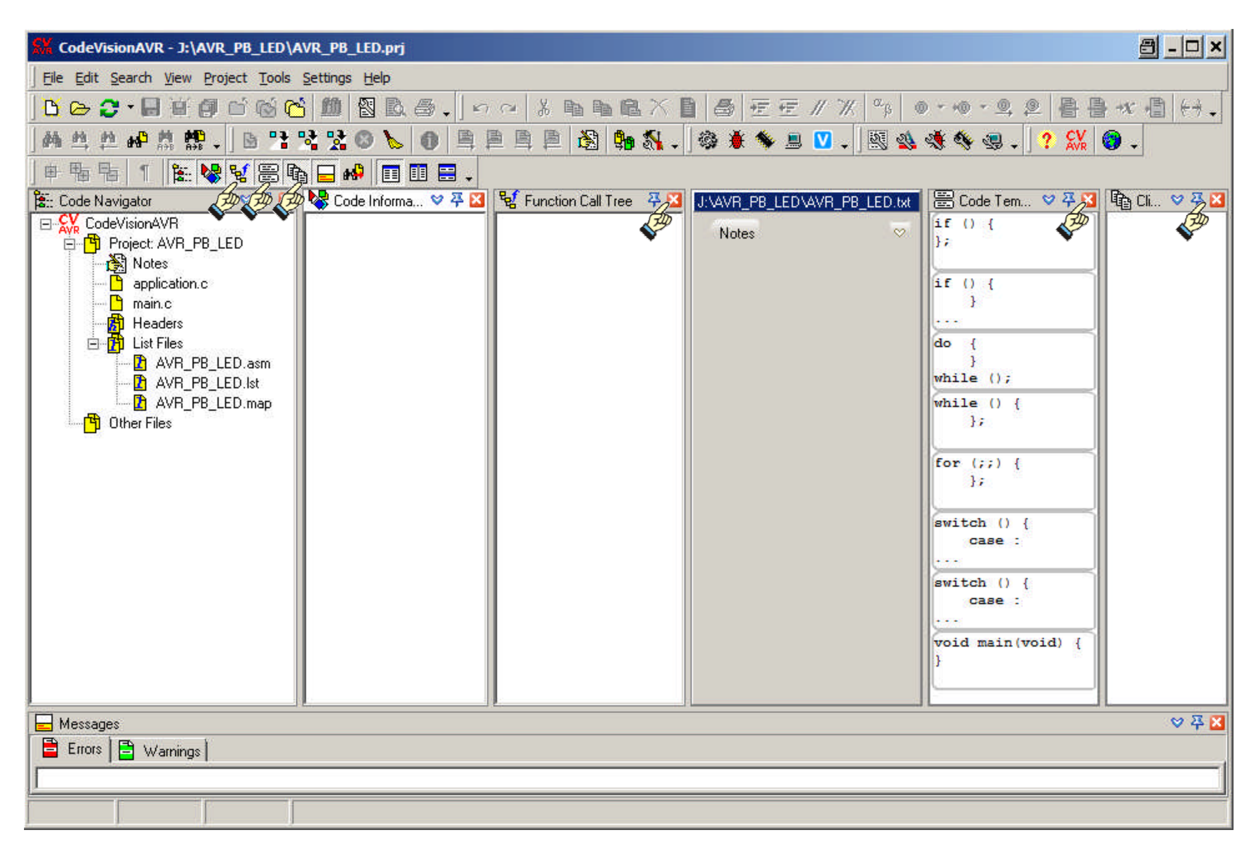


Bild 6.2.1-12: Die Spalten "Function Call Tree", "Code Templates" und "Clipboard History" werden zunächst entfernt, um Platz für die `*.c`-Dateien zu erhalten

In der "File Pane" (Editier-Feld, das ist die 4. Spalte von links) - wo bereits die leere `*.txt`-Datei unter "Notes" vorgemerkt ist - können alle `*.c`-Dateien auch nebeneinander oder untereinander angezeigt werden. In der **`AVR_PB_LED.txt`**-Datei kann das Projekt unter "Notes" detailliert beschrieben werden - das soll uns hier aber noch nicht interessieren.

Wenn die Felder nicht gleich so erscheinen, wie sie im **Bild 6.2.1-12** dargestellt sind, so ist zu bemerken, dass die Felder - wie in anderen Anwendungen auch - minimiert, maximiert oder "gezogen" werden können, was hier geschehen ist.

Um Platz für die Quelldateien zu schaffen, werden jetzt die Spalten **Function Call Tree, Code Templates** und **Clipboard History** zunächst entfernt und die "File Pane" verbreitert:

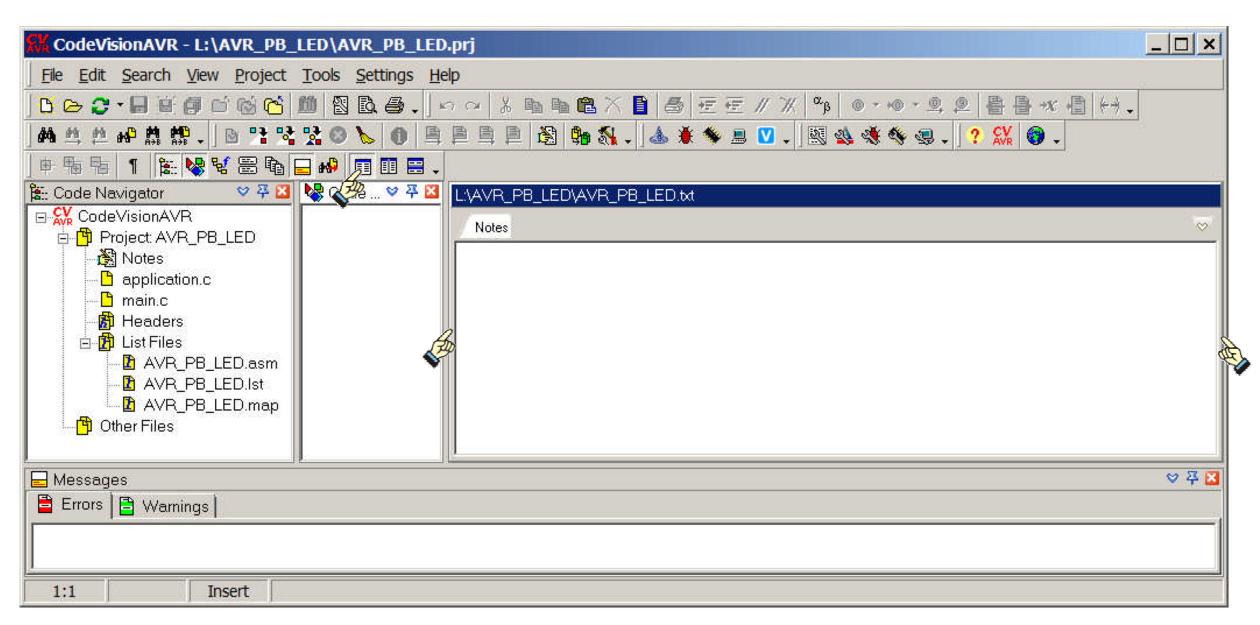


Bild 6.2.1-13: Verbreiterte "File Pane"

Mit ein wenig Jonglieren werden die Spalten gestaltet:

1. Spalten nebeneinander
2. Öffnen von `main.c` im **Code Navigator**
3. Öffnen von `application.c` im **Code Navigator**

Wie man sieht, kann man im großen Fenster noch eifrig alle Quell-Dateien verändern/editieren. Die farbliche Kennzeichnung der Syntaxhervorhebung weicht auch hier vom Beispieltext etwas ab, da der integrierte Editor vom CVAVR für die Syntaxhervorhebung nicht so viele Einstellmöglichkeiten bietet wie der UltraEdit (siehe **Teil 207 Editor - UltraEdit**).

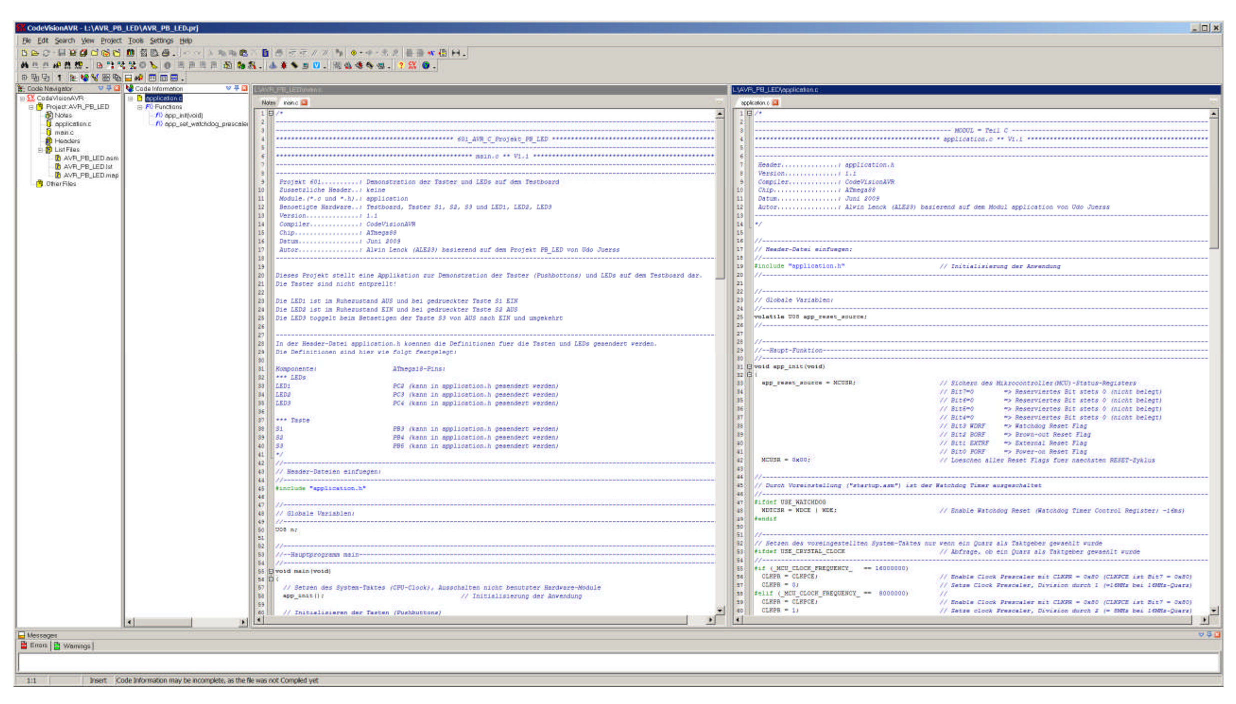


Bild 6.2.1-14: Die `*.c`-Dateien sind bereit zum Editieren (Bildvergrößerung)

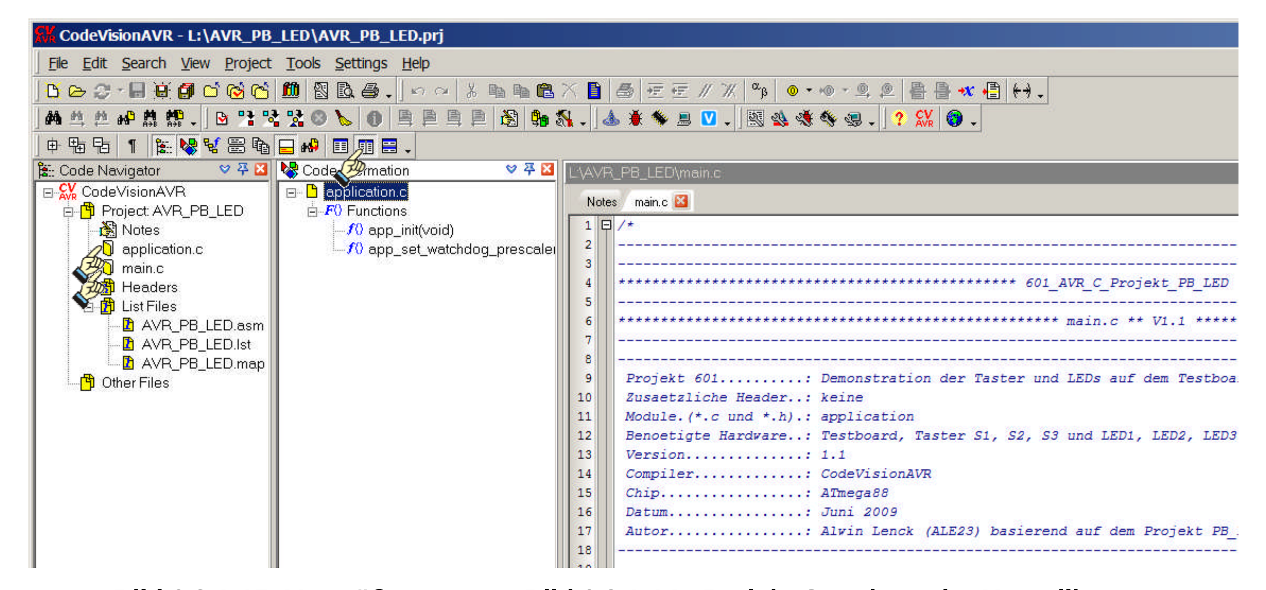


Bild 6.2.1-15: Vergrößerung von Bild 6.2.1-14 - Projekt-Stand vor der Kompilierung

6.2.2 Konfiguration des C-Projektes

Bevor das Programm kompiliert wird, sollen vorher noch einige andere Listen zur Konfiguration des Projektes genauer betrachtet werden. Dazu wird erst einmal das Konfigurations-Menü erneut aufgerufen:

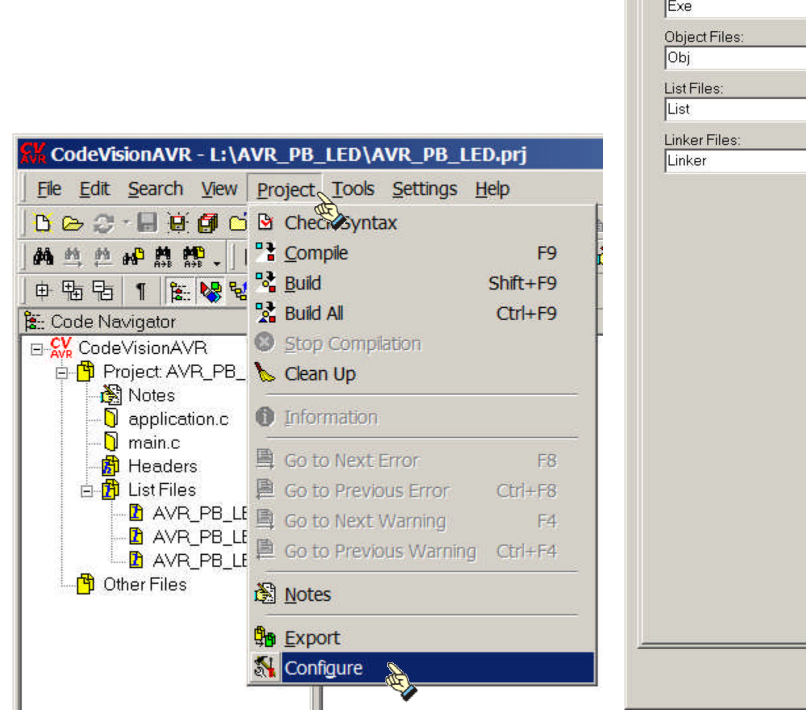


Bild 6.2.2-01: Aufruf des Konfigurations-Menüs

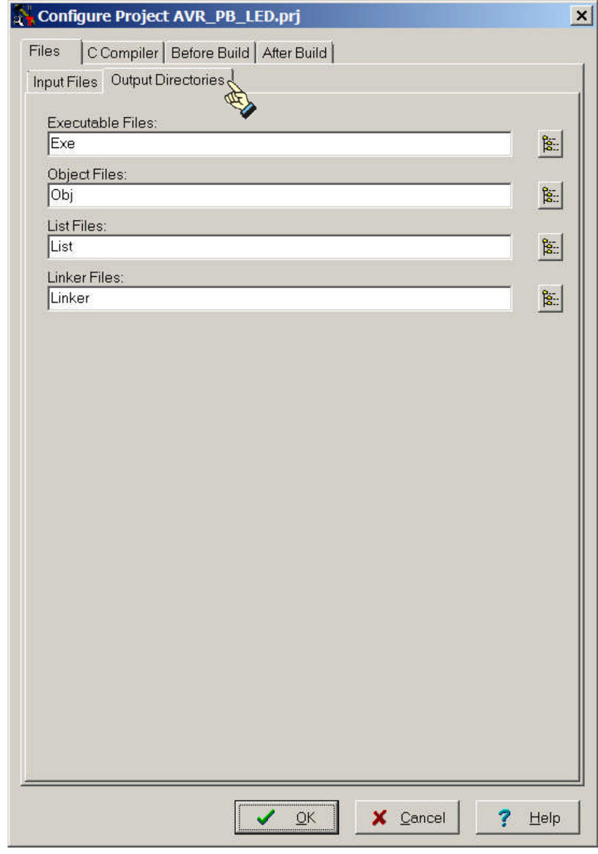


Bild 6.2.2-02: Configure Project Part 2 - Output Directories

Project => Configure => Configure Project AVR_PB_LED.prj => Output Directories

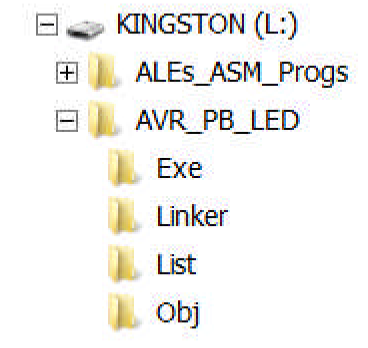
In dieser Abbildung werden die Ausgabe-Ordner angezeigt, in denen bei der Kompilierung die Ergebnis-Dateien abgelegt werden.

Executable Files:
`*.hex`-Dateien vom Linker

Object Files:
`*.obj`-Dateien nach dem Lauf des Assemblers

List Files:
`*.asm`-Dateien nach der Kompilierung

Linker Files:
`*.a`- und `*.o`-Dateien zum Linken/Binden



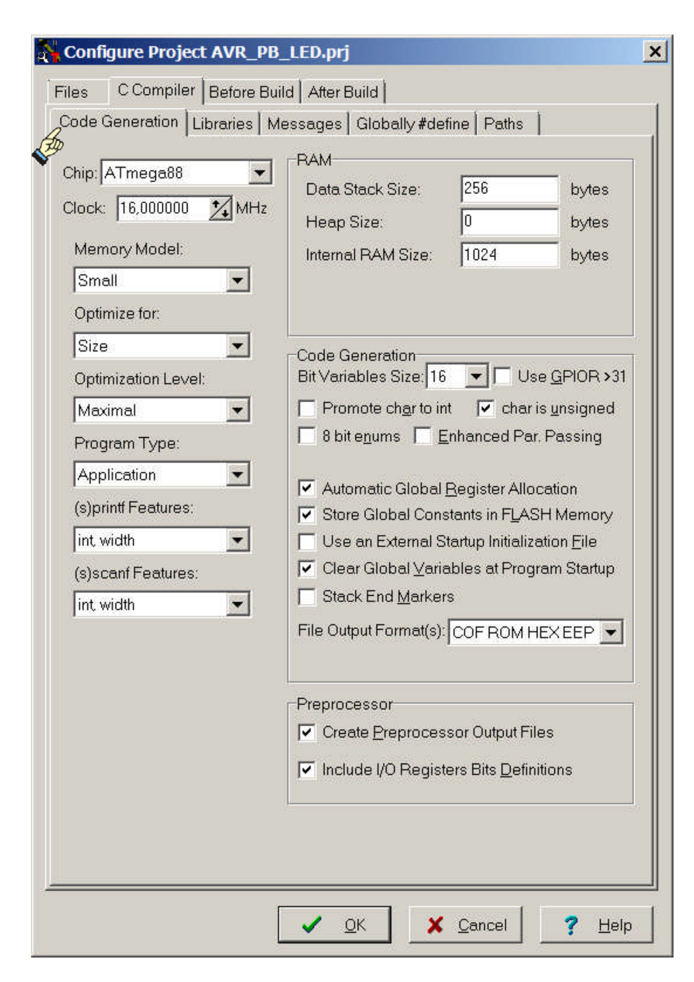


Bild 6.2.2-03: Configure Project Part 3 - Code Generation

Hier können Einstellungen für den Chip und den Compiler zur Generierung der Ausgabe-Dateien vorgenommen werden. Die Einstellungen werden wie angegeben vorgenommen. Besonders zu beachten sind:

Chip:	**ATMega88**
Clock:	**16 MHz**
Program Type:	**Application**
Data Stack Size:	**256 bytes**
Heap Size:	**0 Bytes**
Internal RAM Size:	**1024 bytes**
Bit Variable Size:	**16**
Use GPIOR>31	**nicht gesetzt**

File Output Format(s):
Formate der Ausgabe-Dateien (hier ist vorrangig das Format **HEX** für ein ausführbares Programm von Interesse).

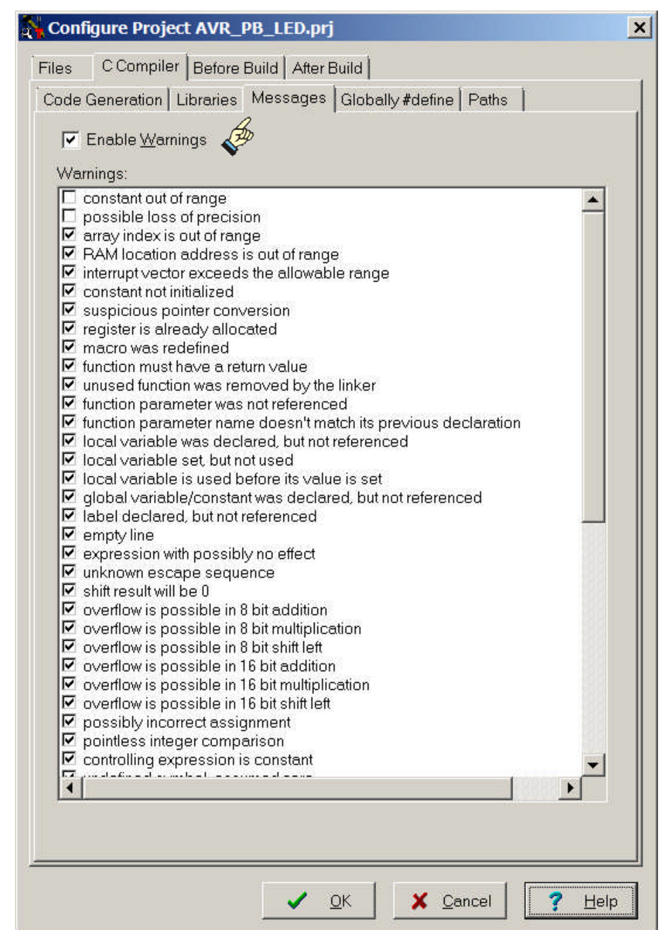


Bild 6.2.2-04: Configure Project Part 4 - Messages

Das ist die Liste **Messages**, in der man angeben kann, welche Hinweise und Warnungen man erhalten möchte.

Das Setzen bestimmter Warnungen kann beim Testen sehr hilfreich sein, um unerreichbaren Code anzuzeigen oder davor zu warnen, dass Variable deklariert sind, die gar nicht benutzt werden.

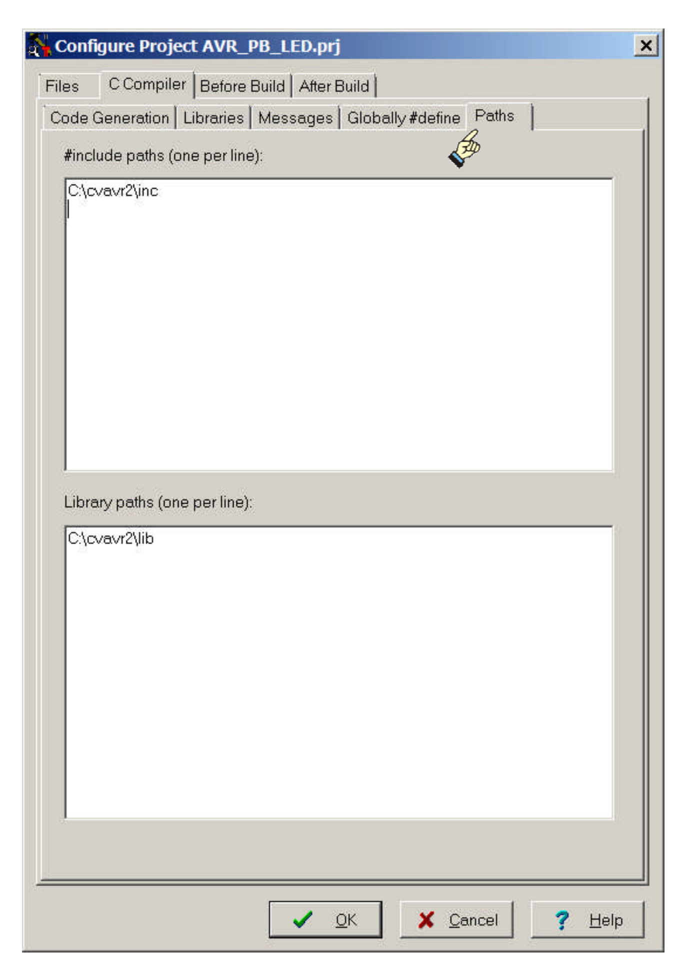


Bild 6.2.2-05: Configure Project Part 5 - Paths

Hier können neue Pfade für zusätzliche `#include`- und library-Dateien angelegt werden:

#include paths (one per line):
für globale Header-Dateien

Library paths (one per line):
für Bibliotheksdateien

Für die hier generierten AVR-Projekte wird von dieser Möglichkeit **kein Gebrauch** gemacht.

6.2.3 Arbeitsschritte zur Generierung eines C-Projektes

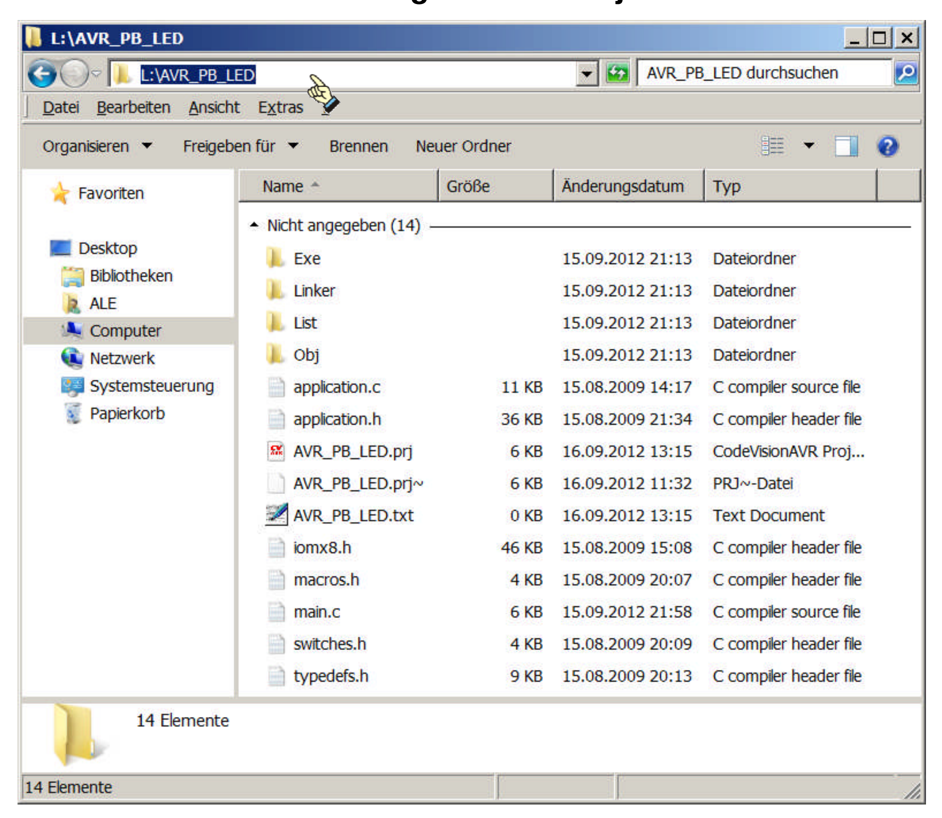


Bild 6.2.3-01: Ausgangslage zur Kompilierung - Stand der Dateien

CodeVisionAVR - L:\AVR_PB_LED\AVR_PB_LED.prj

File Edit Search View Project Tools Settings Help

Code Navigator
- CodeVisionAVR
  - Project: AVR_PB_LED
    - Notes
    - application.c
    - main.c
    - Headers
    - List Files
      - AVR_PB_LED.asm
      - AVR_PB_LED.lst
      - AVR_PB_LED.map
  - Other Files

Code Information
- main.c
  - Functions
    - main(void)

L:\AVR_PB_LED\main.c

Notes | main.c

```c
/*
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
********************************************* 601_AVR_C_Projekt_PB_LED *********************************************
--------------------------------------------------------------------------------
************************************************** main.c ** V1.1 **************************************************
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
 Projekt 601..........: Demonstration der Taster und LEDs auf dem Testboard
 Zusaetzliche Header..: keine
 Module.(*.c und *.h).: application
 Benoetigte Hardware..: Testboard, Taster S1, S2, S3 und LED1, LED2, LED3
 Version..............: 1.1
 Compiler.............: CodeVisionAVR
 Chip.................: ATmega88
 Datum................: Juni 2009
 Autor................: Alvin Lenck (ALE23) basierend auf dem Projekt PB_LED von Udo Juerss
--------------------------------------------------------------------------------

Dieses Projekt stellt eine Applikation zur Demonstration der Taster (Pushbottons) und LEDs auf dem Testboard dar.
Die Taster sind nicht entprellt!

Die LED1 ist im Ruhezustand AUS und bei gedrueckter Taste S1 EIN
Die LED2 ist im Ruhezustand EIN und bei gedrueckter Taste S2 AUS
Die LED3 toggelt beim Betaetigen der Taste S3 von AUS nach EIN und umgekehrt

--------------------------------------------------------------------------------
In der Header-Datei application.h koennen die Definitionen fuer die Tasten und LEDs geaendert werden.
Die Definitionen sind hier wie folgt festgelegt:

Komponente:                    ATmega18-Pins:
*** LEDs
LED1                           PC2 (kann in application.h geaendert werden)
LED2                           PC3 (kann in application.h geaendert werden)
LED3                           PC4 (kann in application.h geaendert werden)

*** Taste
S1                             PB3 (kann in application.h geaendert werden)
S2                             PB4 (kann in application.h geaendert werden)
S3                             PB5 (kann in application.h geaendert werden)
*/
//------------------------------------------------------------------------------
// Header-Dateien einfuegen:
//------------------------------------------------------------------------------
#include "application.h"

//------------------------------------------------------------------------------
// Globale Variablen:
//------------------------------------------------------------------------------
U08 n;

//------------------------------------------------------------------------------
//--Hauptprogramm main----------------------------------------------------------
//------------------------------------------------------------------------------
void main(void)
{
  // Setzen des System-Taktes (CPU-Clock), Ausschalten nicht benutzter Hardware-Module
  app_init();                                          // Initialisierung der Anwendung

  // Initialisieren der Tasten (Pushbuttons)
  S1_INIT();
  S2_INIT();
  S3_INIT();

  // Initialisieren der LEDs
  LED1_DDR |= LED1_BIT;                                // Setze LED1-Pin auf Ausgabe
  LED2_DDR |= LED2_BIT;                                // Setze LED2-Pin auf Ausgabe
  LED3_DDR |= LED3_BIT;                                // Setze LED3-Pin auf Ausgabe

  // Alle LEDs 3-mal aufleuchten lassen ("Lampen-Test")
  for (n = 3; n; n--)
  {
    LED1_PRT |= LED1_BIT;                              // LED1 auf EIN
    LED2_PRT |= LED2_BIT;                              // LED2 auf EIN
    LED3_PRT |= LED3_BIT;                              // LED3 auf EIN

    delay_ms(250);                                     // 250 ms Verzoegerung (1/4 Sekunde)

    LED1_PRT &= ~LED1_BIT;                             // LED1 auf AUS
    LED2_PRT &= ~LED2_BIT;                             // LED2 auf AUS
    LED3_PRT &= ~LED3_BIT;                             // LED3 auf AUS

    delay_ms(200);                                     // 200 ms Verzoegerung (1/5 Sekunde)
  }

//------------------------------------------------------------------------------
// Hauptschleife:
//------------------------------------------------------------------------------
  while (true)                                         // Endlosschleife
  {
    // Wenn Taste S1 gedrueckt wird, dann leuchtet LED1
    if (S1_PRESSED())
    {
      LED1_PRT |= LED1_BIT;                            // LED1 auf EIN
    }
    else
    {
      LED1_PRT &= ~LED1_BIT;                           // LED1 auf AUS
    }

    // Wenn Taste S2 gedrueckt wird, dann schaltet LED2 auf AUS
    if (S2_PRESSED())
    {
      LED2_PRT &= ~LED2_BIT;                           // LED2 auf AUS
    }
    else
    {
      LED2_PRT |= LED2_BIT;                            // LED2 auf EIN
    }

    // Wenn Taste S3 gedrueckt wird, dann toggle LED3 (EIN-AUS-EIN-AUS-...)
    if (S3_PRESSED())
    {
      LED3_PRT ^= LED3_BIT;                            // Toggle LED3

      S3_WAIT_NOT_PRESSED();                           // Warte bis Taste S3 losgelassen wird
    }
  }
}
//------------------------------------------------------------------------------
```

Messages

Errors | Warnings

119:98 | Insert | Code Information may be incomplete, as the file was not Compiled yet

Bild 6.2.3-02: Ausgangslage zur Kompilierung - CVAVR (Bildvergrößerung)

Die Ausgangslage für die Kompilierung sind in einem **C**-Projekt die sog. Quell-Dateien. In diesem Projekt sind das:

- Das Hauptprogramm **main.c**
- Das Modul **application.c** incl. **application.h**
- Die Header-Datei **typedefs.h**
- Die Header-Datei **iomx8.h**
- Die Header-Datei **macros.h**
- Die Header-Datei **switches.h**

Aus ihnen entsteht in grundsätzlich 4 Arbeitsschritten das fertige "Kompilat", das heißt die ausführbare ***.hex**-Datei:

1. Arbeitsschritt "Lauf des Preprozessors": Auswerten aller Anweisungen, die mit # beginnen. Das sind Anweisungen, die ausschließlich für den Preprozessor bestimmt sind (u. a. Einfügen von Header-Dateien, die mit **#include "*file*"** in den Quell-Dateien aufgeführt sind.

2. Arbeitsschritt "Lauf des C-Compilers" zur Übersetzung der in der Programmiersprache **C** geschriebenen Befehle in Assembler-Anweisungen **=>** ***.asm**-Dateien (dieses sind ebenfalls editierbare Text-Dateien).

3. Arbeitsschritt "Lauf des Assemblers" zur Übersetzung der Assembler-Anweisungen in Maschinen-Befehle (Module im Objekt-Code) **=>** ***.a**-Dateien und ***.o**-Dateien

4. Arbeitsschritt "Lauf des Linkers (Binders)" zur Verbindung aller Objekt-Module zu einem lauffähigen ausführbaren Computer-Programm **=>** ***.hex**-Datei

In **CVAVR** kann das **C**-Projekt sequentiell, d.h. in Einzel- bzw. Stufenschritten, generiert werden:

Project => Compile (F9)

Dieser Schritt produziert die Objekt-Dateien für den Linker. Die Kompilierung wird nur für die nach dem letzten Durchlauf veränderten Programm-Module durchgeführt.

Project => Build (Shift+F9)

Dieser Schritt produziert ein neues Assembler-Quell-Programm mit der Erweiterung ***.asm** wobei nur die veränderten Quell-Dateien neu kompiliert werden. Wenn keine Fehler erkannt wurden, wird automatisch der **Atmel-AVR-Assembler** aufgerufen, der die eben erzeugte **.asm**-Datei assembliert. Das Ausgabe-File ist eine ausführbare Programm-Datei mit dem Format ***.hex**.

Project => Build All (Ctrl+F9)

Vollständige Kompilierung/Assemblierung/Verlinkung aller Quellen; Zusammenfassung aller Arbeitsschritte zu einem Arbeitsgang.

Da das Projekt bereits an anderer Stelle getestet wurde, sind keine Fehler (mehr) zu erwarten, so dass sofort mit dem Schritt **Build All (Ctr+F9)** fortgesetzt wird:

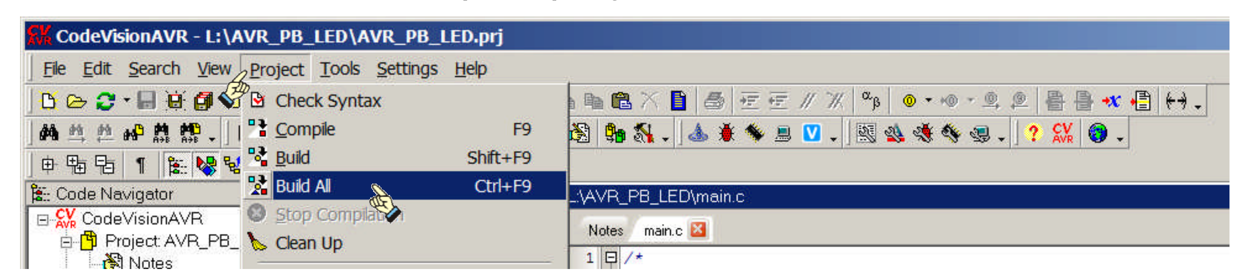


Bild 6.2.3-03: Project => Build All (Ctrl+F9)

Wenn die vollständige Kompilierung fehlerfrei (aber wahrscheinlicht nicht kommentarlos !) gelaufen ist, werden über dem Haupt-Fenster von CVAVR zwei Informations-Listen am Ende der Projekt-Generierung offeriert. Sie zeigen die Ergebnisse für die Kompilierung und die Assemblierung.

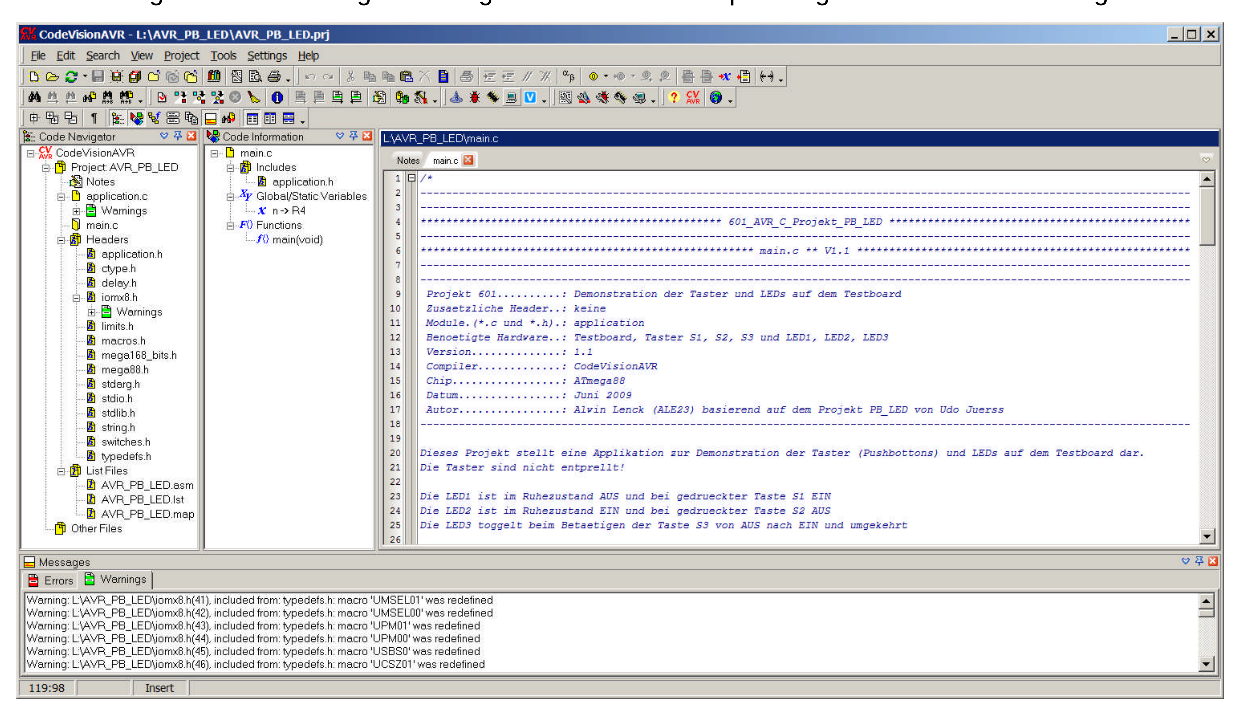


Bild 6.2.3-04: Nach der Kompilierung mit Build All (Ctrl+F9) (Bildvergrößerung)

Information

Compiler | Assembler

Chip: ATmega88
Clock frequency: 16,000000 MHz
Program type: Application
Memory model: Small
Optimize for: Size
(s)printf features: int, width
(s)scanf features: int, width
Promote 'char' to 'int': No
'char' is unsigned: Yes
global 'const' stored in FLASH: Yes
8 bit enums: No
Enhanced function parameters passing: No
Automatic register allocation: Yes
Smart register allocation: Yes

Build: 1
15441 line(s) compiled
No errors
604 warnin[gs]

Bit variables size: 0 byte(s)

Data Stack area: 0x100 to 0x1FF
Data Stack size: 256 byte(s)
Estimated Data Stack usage: 0 byte(s)

RAM Global variables area: 0x200 to 0x204
RAM Global variables size: 5 byte(s)

Hardware Stack area: 0x205 to 0x4FF
Hardware Stack size: 763 byte(s)

Heap size: 0 byte(s)

EEPROM usage: 0 byte(s), 0,0% of EEPROM
Program size: 182 words (364 bytes), 4,4% of FLASH
0 words (0 bytes), 0,0% reduced by Code Compression

OK

Bild 6.2.3-05: Compiler-Meldungen

Information

Compiler | Assembler

AVRASM: AVR macro assembler 2.1.30 (build 592 Nov 7 2008 12:38:17)
Copyright (C) 1995-2008 ATMEL Corporation

L:\AVR_PB_LED\List\AVR_PB_LED.asm(1659): No EEPROM data, delet

ATmega88 memory use summary [bytes]:

Segment	Begin	End	Code	Data	Used	Size	Use%
[.cseg]	0x000000	0x00016c	326	38	364	8192	4.4%
[.dseg]	0x000100	0x000205	0	5	5	1024	0.5%
[.eseg]	0x000000	0x000000	0	0	0	512	0.0%

Assembly complete, 0 errors.

OK

Bild 6.2.3-06: Assembler-Meldungen

Wenn man tiefer in die Materie eingestiegen ist, so ergeben diese Listen sehr aufschlussreiche Hinweise zum Kompilat und den benutzten Ressourcen (Speicher-Belegungen). An dieser Stelle mögen die Hinweise

No errors und **Assembly complete, 0 errors.**

genügen. Sie zeigen an, dass das Projekt fehlerfrei kompiliert und assembliert wurde und ein ablauffähiges Objekt-Programm erzeugt wurde.

6.3 Einbinden von AVR Studio in den CVAVR

Der Compiler CVAVR ist dafür vorgesehen, mit dem Debugger vom AVR Studio ab der Version 4.13 zu kommunizieren. Dazu ist es notwendig, die Lokalisation von AVR Studio, d.h. den Ordner-Pfad von AVR Studio auf der System-Platte, der Anwendung CVAVR bekannt zu machen. Diese Lokalisation ist (unter der Annahme, dass AVR Studio auf seinem "angestammten" Platz installiert wurde):

```
C:\Program Files (x86)\Atmel\AVR Tools\AvrStudio4\AVRStudio.exe
```

Der Eintrag in CVAVR wird im Menü **Debugger Settings** vorgenommen:

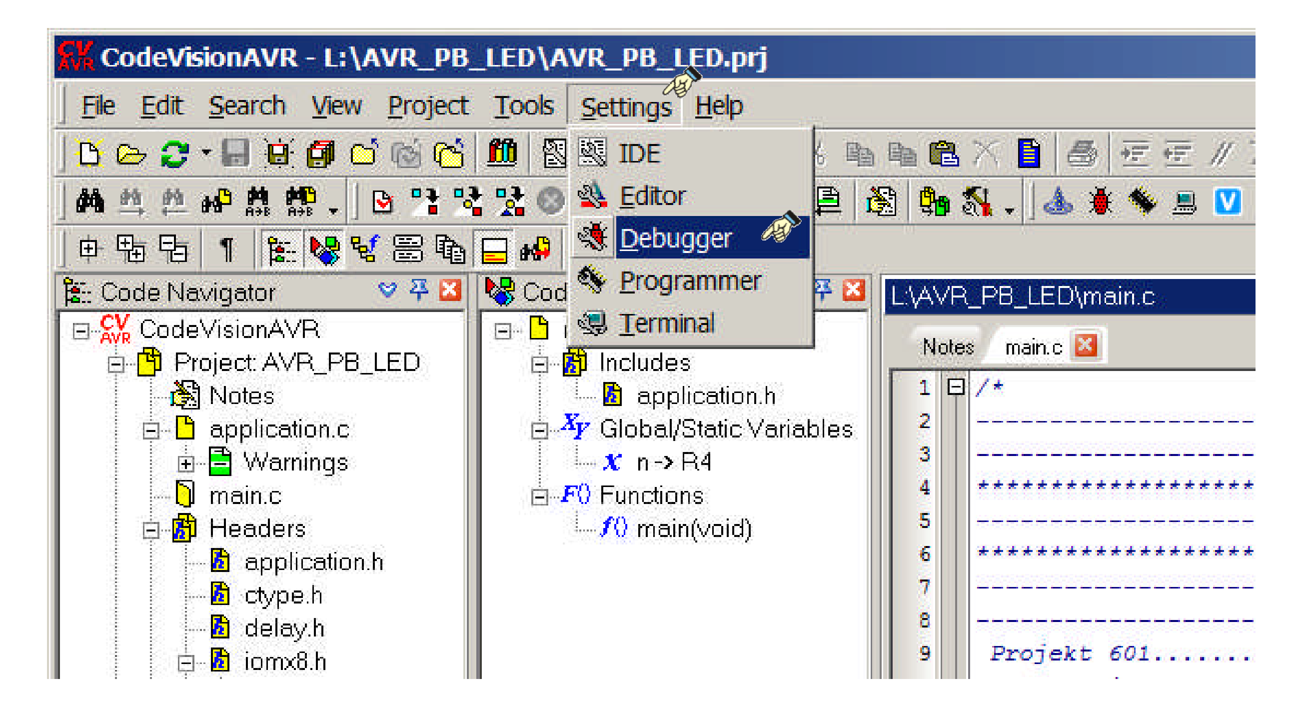


Bild 6.3-01: Debugger-Eintrag in CVAVR vornehmen

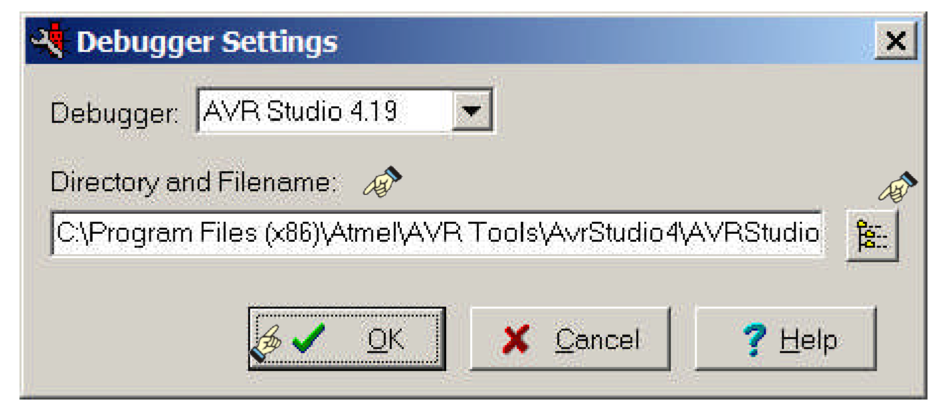


Bild 6.3-02: Einstellen der Datei AVRStudio.exe

Settings => Debugger => Debugger Settings => Directory and Filename:
`C:\Program Files (x86)\Atmel\AVR Tools\AvrStudio4\AVRStudio.exe`
=> OK

Sollte **AVRStudio.exe** nicht an der vorgesehenen Lokalisation installiert worden sein, so kann sie mit der "File-Such-Funktion" gefunden werden:

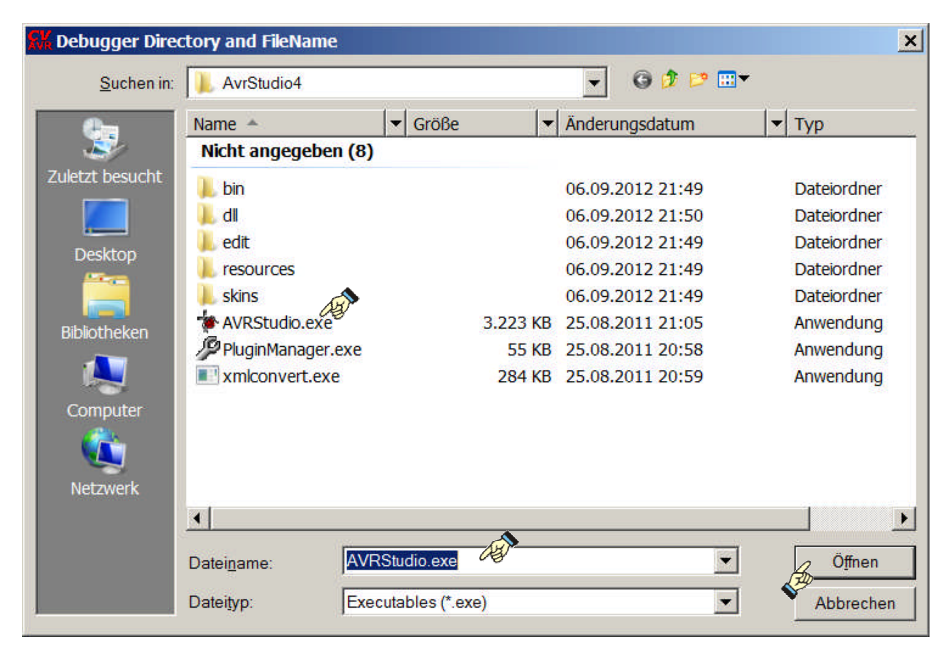


Bild 6.3-03: Suchen der Datei AVRStudio.exe

Hinweis: Hier findet ein gleitender Übergang zum nächsten Abschnitt statt. Man befindet sich scheinbar noch in der Anwendung **CVAVR**. Denn aus dieser heraus wird der **Debugger** aufgerufen, der aber schon Bestandteil des **AVR Studio** ist.

6.4 AVR Studio Debugger

Mit dem Debugger des AVR Studios kann man - bevor man den Objekt-Code des fertig kompilierten Projektes in den Flash-Speicher des Mikrocontrollers überträgt - Schritt für Schritt den Programmablauf simulieren.

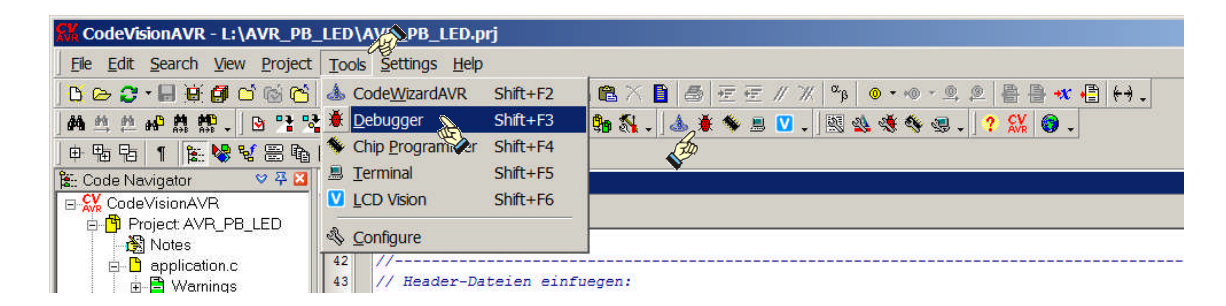


Bild 6.4-01: Aufruf des Debuggers zum Projekt AVR_PB_LED

Tools => Debugger (Shift+F3) => öffnet AVR Studio => File => Open File...(Strg+O)

Es startet AVR Studio (aus CVAVR heraus !):

Dabei wird ggf. das Startfenster mit einem Willkommensdialog geöffnet, wenn die Option **Show this dialog on open** aktiviert war. Da nicht mit **Assembler** oder **AVR GCC** gearbeitet wird, sondern ein AVR-Projekt mit **CVAVR** erstellt werden soll, wird das Häkchen bei **Show dialog at startup** entfernt - und das Fensterchen erscheint zukünftig nicht mehr.

Es ist zu beachten, dass bereits einige Optionen voreingestellt worden sein können, so dass das Abbild von AVR Studio nach der ursprünglichen Installation anders ausgesehen haben könnte.

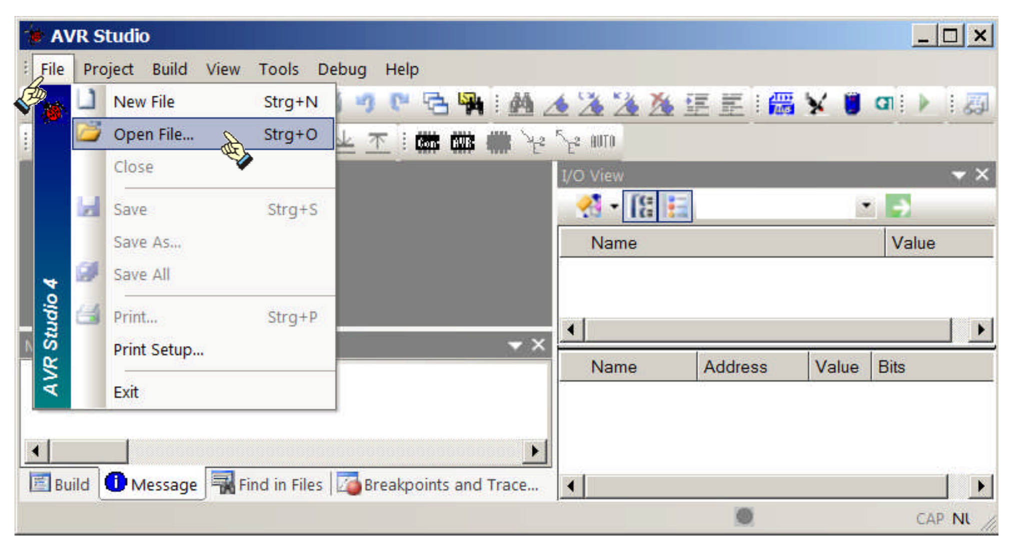


Bild 6.4-02: Öffnen des Datei-Dialogs

Öffnen

L:\AVR_PB_LED | AVR_PB_LED durchsu...

Organisieren ▾ Neuer Ordner

Name	Größe	Änderungsdatum	Typ
Nicht angegeben (25)			
Exe		23.09.2012 19:39	Dateiordner
Linker		23.09.2012 19:39	Dateiordner
List		23.09.2012 19:39	Dateiordner
Obj		23.09.2012 19:39	Dateiordner
application.c	11 KB	15.08.2009 14:17	C compiler source file
application.cci	1 KB	23.09.2012 19:49	CCI-Datei
application.h	36 KB	15.08.2009 21:34	C compiler header file
application.hci	1 KB	23.09.2012 19:49	HCI-Datei
application.i	9 KB	23.09.2012 19:49	C compiler library file
AVR_PB_LED.cbi	1 KB	23.09.2012 19:49	CBI-Datei
AVR_PB_LED.cof	4 KB	23.09.2012 19:49	COFF symbolic debug file
AVR_PB_LED.fct	1 KB	23.09.2012 19:49	FCT-Datei
AVR_PB_LED.prj	6 KB	23.09.2012 19:50	CodeVisionAVR Project File
AVR_PB_LED.prj~	5 KB	23.09.2012 19:40	PRJ~-Datei
iomx8.h	46 KB	15.08.2009 15:08	C compiler header file
iomx8.hci	11 KB	23.09.2012 19:49	HCI-Datei
macros.h	4 KB	15.08.2009 20:07	C compiler header file
macros.hci	1 KB	23.09.2012 19:49	HCI-Datei
main.c	6 KB	15.09.2012 21:58	C compiler source file
main.cci	1 KB	23.09.2012 19:49	CCI-Datei
main.i	9 KB	23.09.2012 19:49	C compiler library file
switches.h	4 KB	15.08.2009 20:09	C compiler header file
switches.hci	1 KB	23.09.2012 19:49	HCI-Datei
typedefs.h	9 KB	15.08.2009 20:13	C compiler header file
typedefs.hci	4 KB	23.09.2012 19:49	HCI-Datei

Dateiname: AVR_PB_LED.cof | All Files (*.*)

Öffnen | Abbrechen

Bild 6.4-03: Suchen der COFF-Datei

Die benötigte **COFF**-Datei befindet sich im Verzeichnis **`L:\AVR_PB_LED`** des Projekt-Ordners vom Beispiel-Projekt **AVR_PB_LED** und hierher soll auch die Datei **`AVR_PB_LED.cof.aps`** abgespeichert werden:

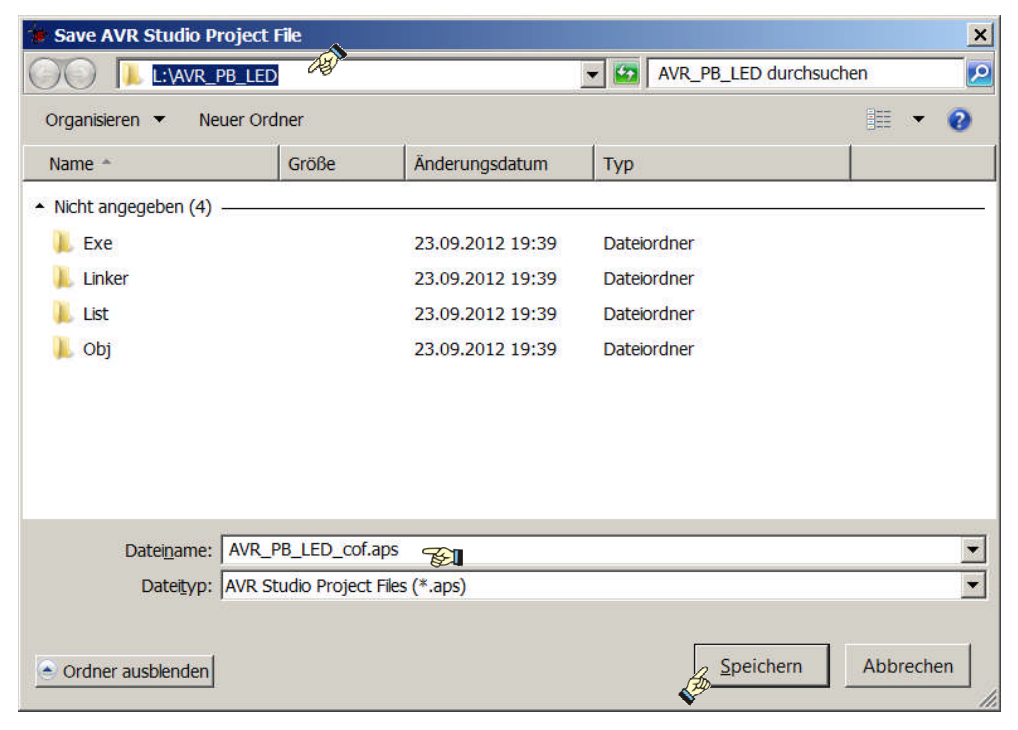


Bild 6.4-04: `AVR_PB_LED.cof.aps` speichern

Es dauert einen Moment, bis man die Debug-Plattform **AVR Simulator** für den Mikrocontroller **ATmega88** einstellen kann:

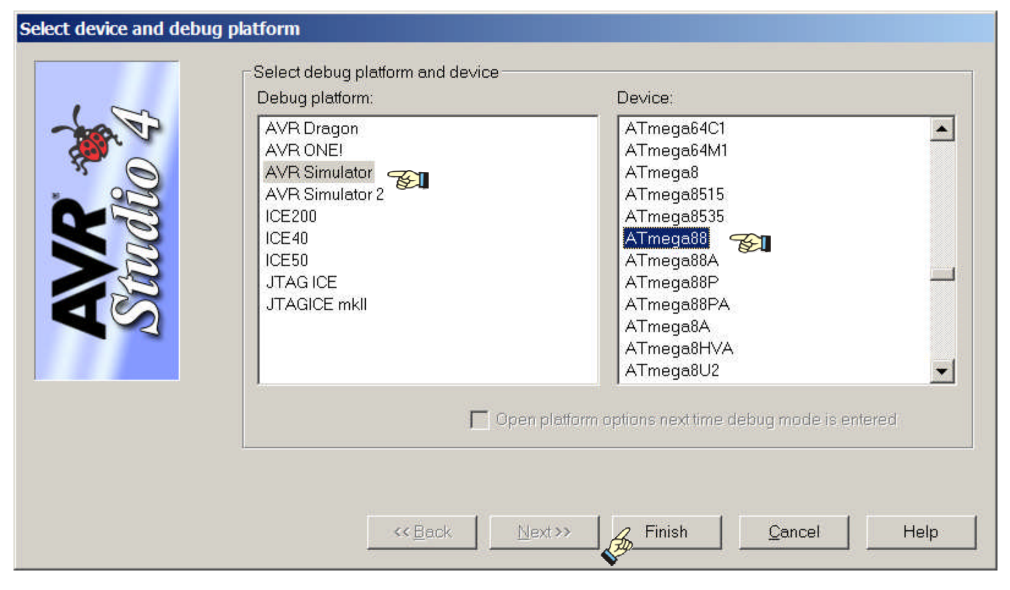


Bild 6.4-05: Debug-Plattform für den ATmega88 einstellen

Nach einer weiteren kurzen Verweildauer öffnet sich AVR Studio mit der für das C-Projekt **AVR_PB_LED** eingestellten Debug-Plattform mit zahlreichen **Toolbars**, von denen besonders hervorzuheben sind:

- Prozessor-Daten (Counter, Pointer, Register usw.)
- Die aktuelle Quell-Datei **`D:\AVR_PB_LED\main.c`** mit einem gelben Pfeil auf die erste ausführbare Anweisung
- Ein-/Ausgabe-Werte (I/O View; Hardware-Ressourcen)
- Nachrichten über die laufenden Aktivitäten (Message Output)

Jetzt "schiebt" man sich die Seiten der Felder so hin, dass eine vernünftige Übersicht erscheint:

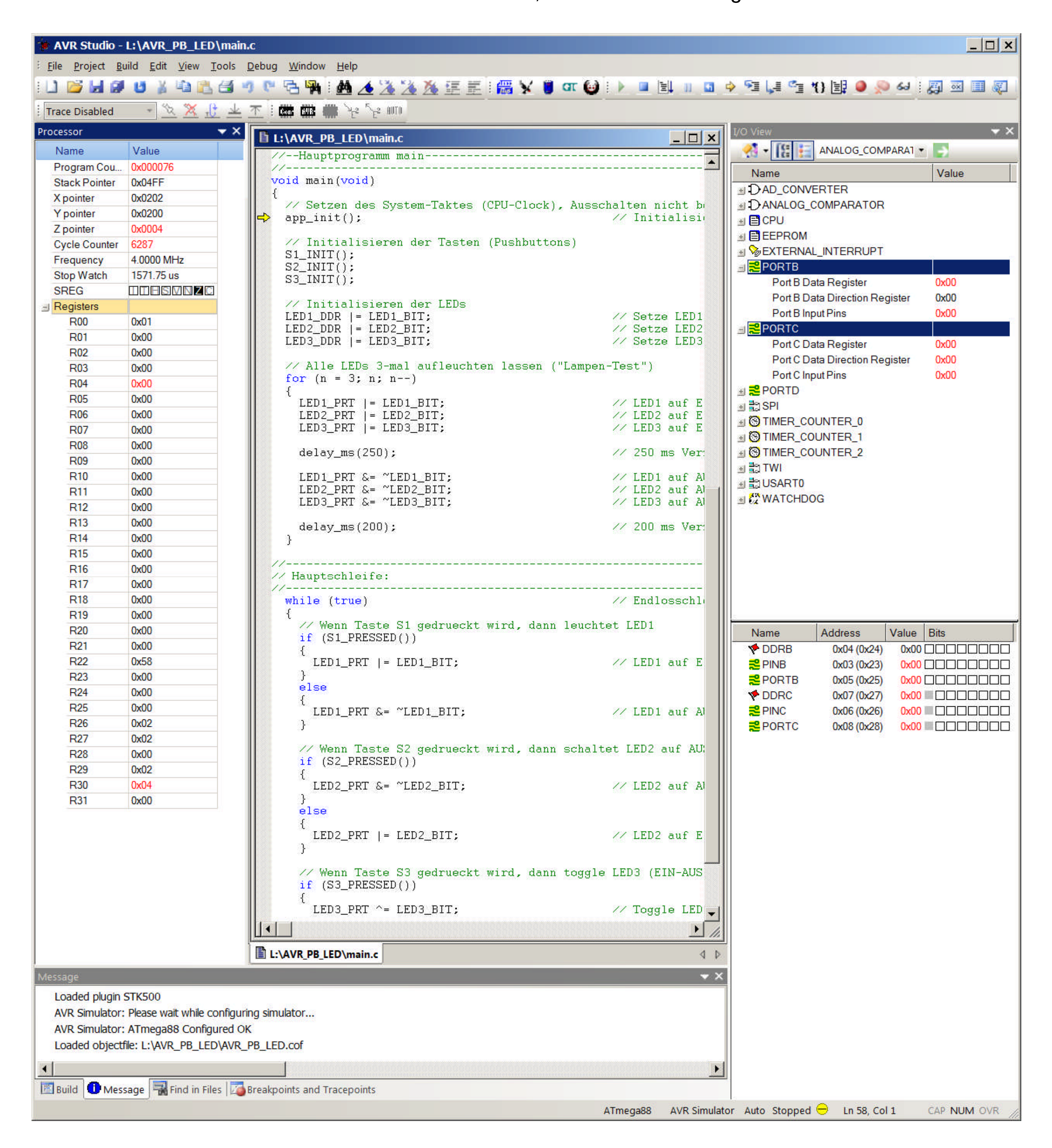


Bild 6.4-06: Debug-Plattform mit Quell-Datei `main.c` (Bildvergrößerung)

Mit der Funktionstaste **F10** bzw. **F11** vom **PC** kann man nun Schritt für Schritt den Programmablauf simulieren. Wenn man dabei den Cursor auf eine Variable positioniert, dann bekommt man den jeweiligen Wert angezeigt. Im Menüpunkt **Debug** kann man sehen was sonst noch so alles möglich ist. Hier nur ein paar Hinweise der zahlreichen Möglichkeiten des Debuggers:

Wenn man in CVAVR eine Änderung im Quell-Programm vornimmt, neu kompiliert und wieder nach AVR Studio wechselt, dann merkt AVR Studio, dass sich die `*.cof`-Datei geändert hat und fragt nach, ob diese Datei neu geladen werden soll. Wenn diese Anfrage mit **JA** beantwortet wird, dann startet der Debugger nach dem Neuladen wieder von vorne.

Wenn AVR Studio neu aus CVAVR heraus gestartet wird, dann kann ein vorhandenes Projekt unter dem Menüpunkt **Project => Recent Project** ***`Projektname`*** wieder unkompliziert geladen werden.

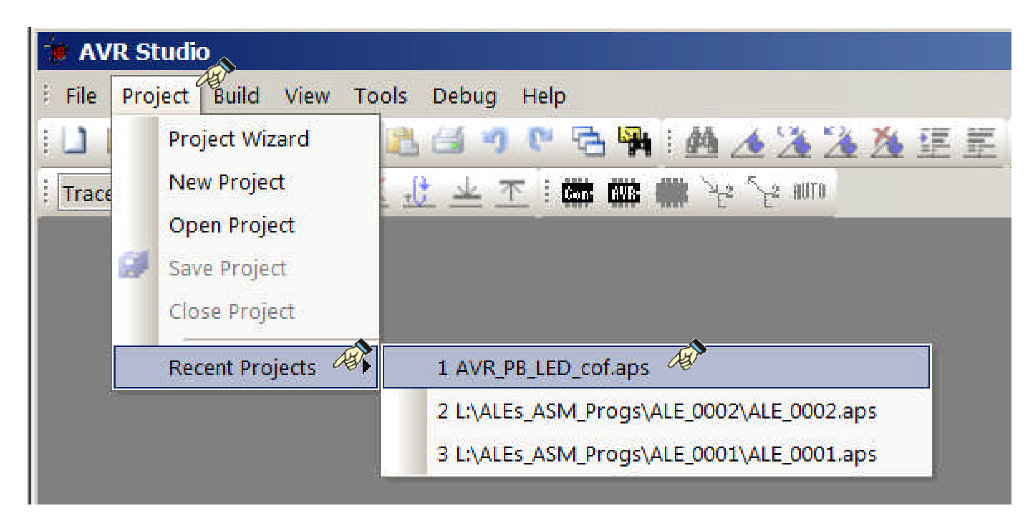


Bild 6.4-07: Fortsetzen des Debugging nach einem Neustart

Wenn man in diesem Projekt beispielsweise auf **PORTC** klickt, dann kann man während der schrittweisen Programmausführung (mit **F10**) sehen, wie sich **PC2**, **PC3**, **PC4** (entsprechend dem kurzen Aufleuchten der LED's) nach dem Reset verändern.

6.5 Flashen eines C-Programms in ein Mikrocontroller ATmega88

Das Programm AVR_PB_LED ist jetzt kompiliert und simuliert worden. Aber ein Test im "Wirkbetrieb" ist natürlich noch wichtiger. Dazu muss der Objektcode (das kompilierte Programm) in den Mikrocontroller (hier in den Mikrocontroller auf dem AVR-ALE-Testboard) geflasht werden und es müssen die notwendigen Verbindungen zwischen den Pins hergestellt werden. Die Beschaltung ist identisch mit dem Abschnitt **1.2 Beschaltung** im **Teil 601 - AVR_PB_LED**:

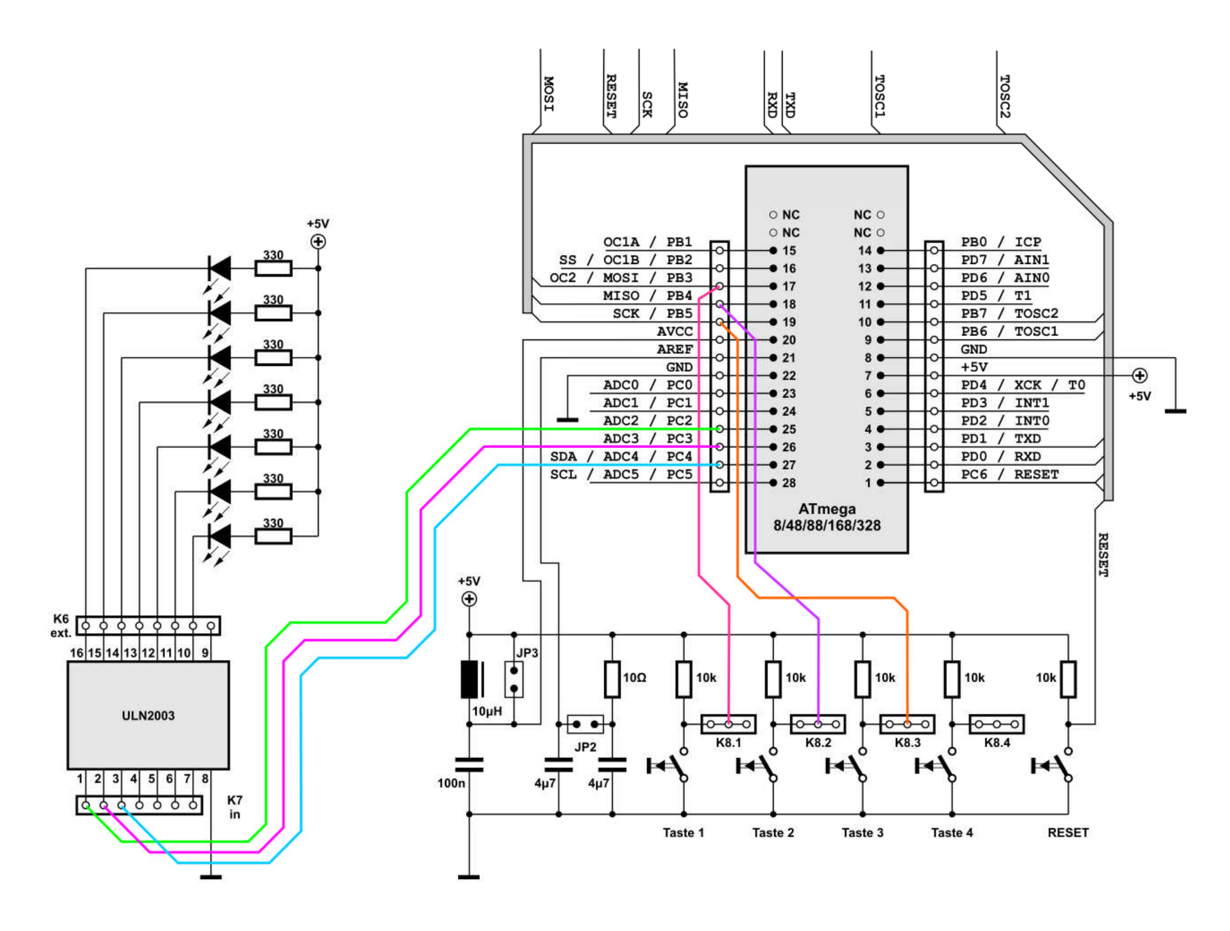


Bild 6.5-01: Beschaltung der LEDs und Taster (Bildvergrößerung)

Dann wird das Board mit seinem Netzteil mit Spannung versorgt.

Der ISP-Programmieradapter wird mit seinem USB-Stecker in einen freien USB-Steckplatz des PCs gesteckt, in dem das neue Programm abgespeichert wurde.

Überflüssige Schaltkreise und die Stromversorgung wurden in dem Beschaltungsbild weggelassen. AVR Studio wird nun erneut gestartet.

Dann wird die logische Verbindung zum **CC2-AVR Programmer** (ISP-Programmieradapter) hergestellt. Die Einstellungen wurden ja schon im Abschnitt **4.3.2 Mikrocontroller-Einstellungen im AVR Studio** von **Teil 204 AVR Studio** vorgenommen, so dass man sich damit nicht mehr aufhalten muss:

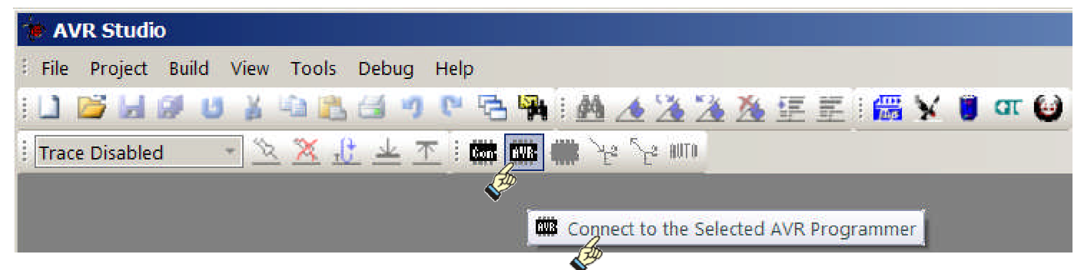


Bild 6.5-02: Mit dem CC2-AVR-Programmer eine Verbindung herstellen

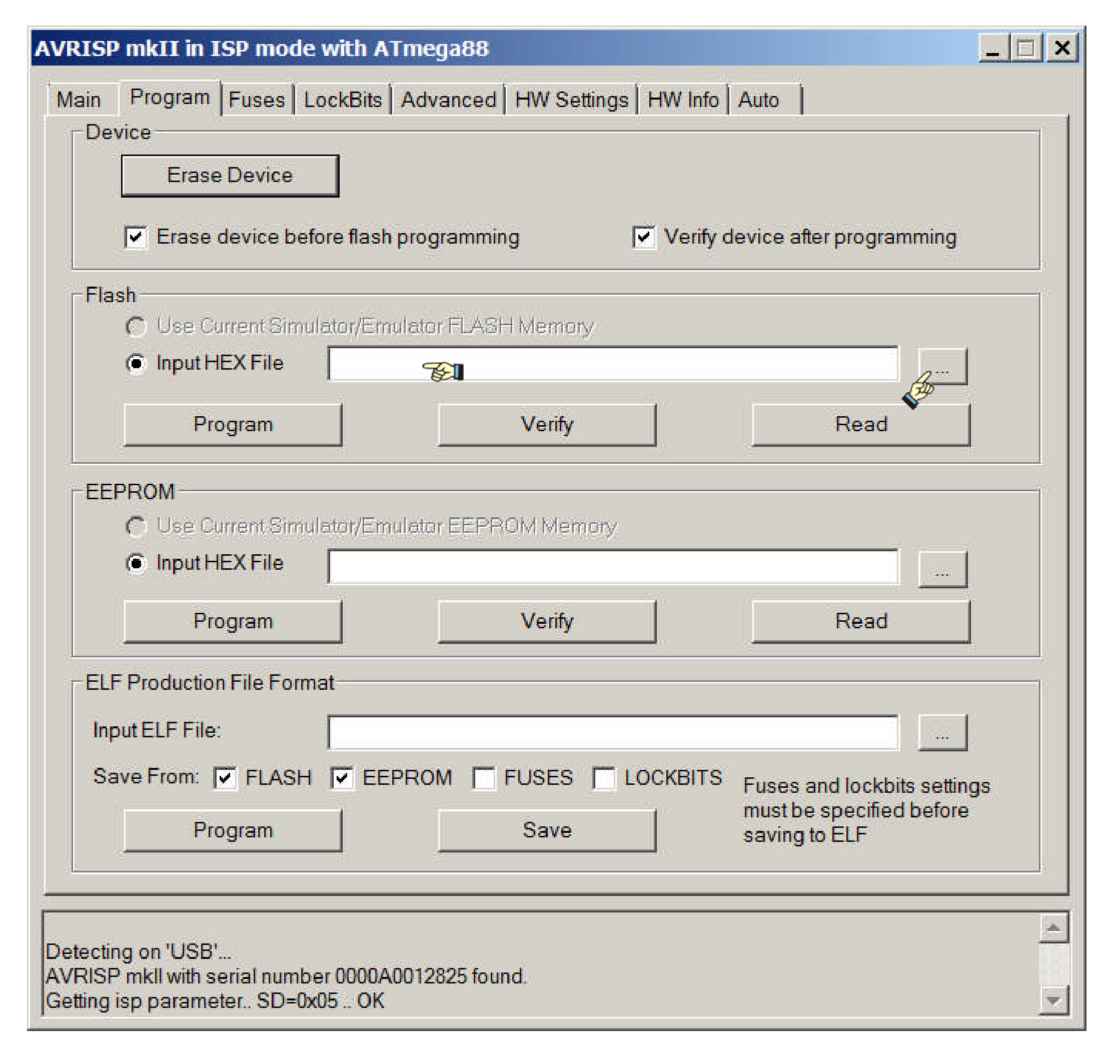


Bild 6.5-03: Objektdatei (*.hex-File) suchen

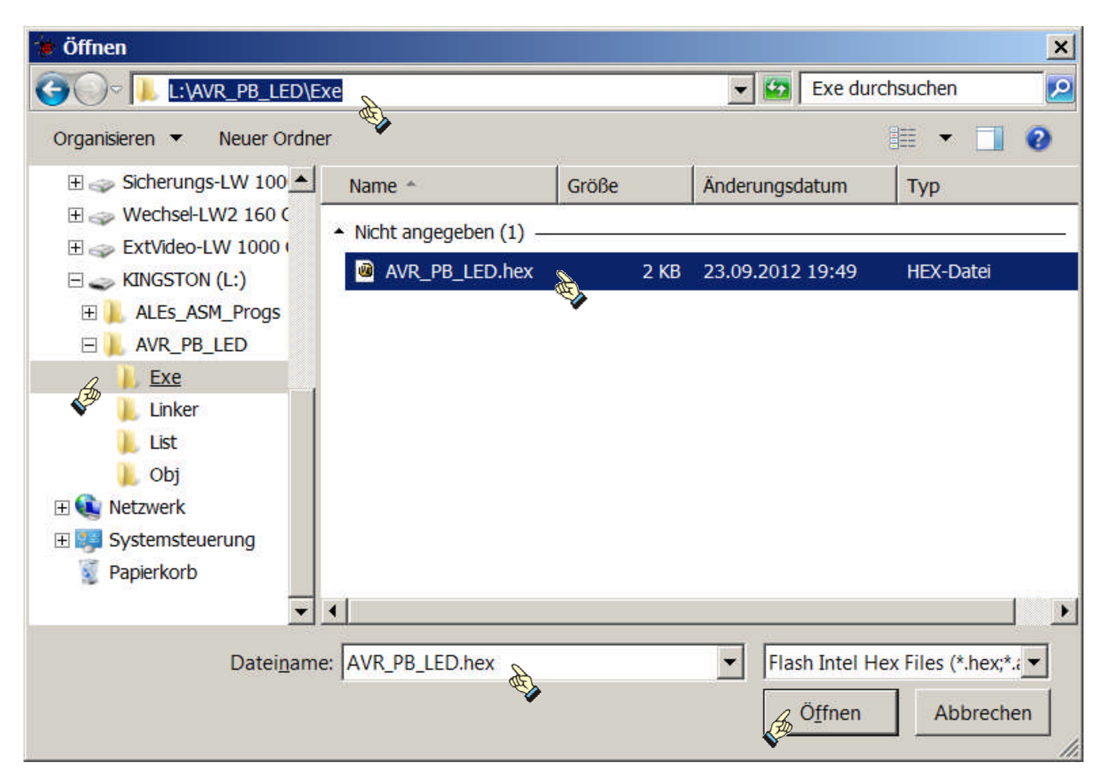


Bild 6.5-04: Objektdatei öffnen

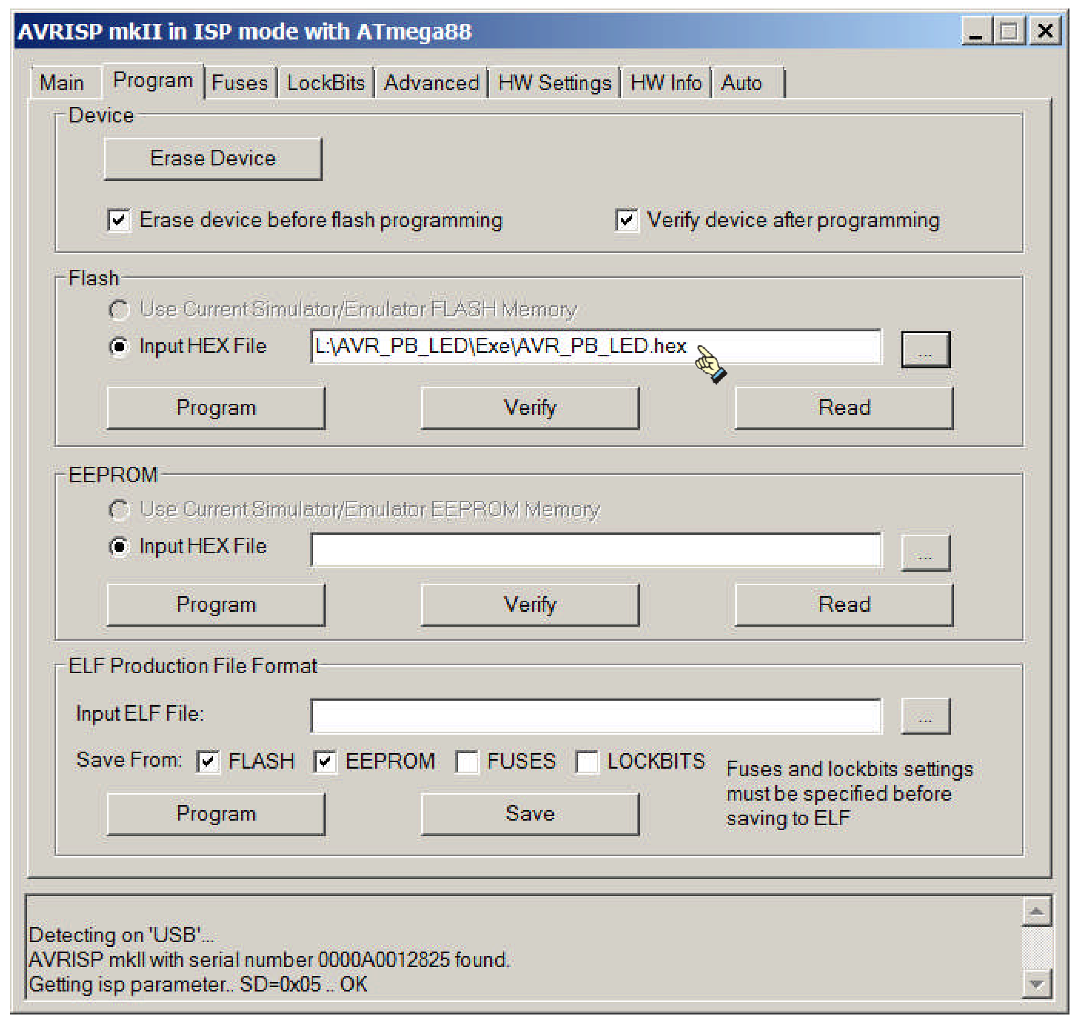


Bild 6.5-05: Programm flashen ==> und das Programm testen ==> alles O.K.